The Complexity of Verifying Feedforward Neural Networks in Quantised Settings

Eric Alsmann[1] , **Martin Lange**[1] , **Marco Sälzer**[2]
[1]University of Kassel
[2]RPTU University Kaiserslautern-Landau
{eric.alsmann, martin.lange}@uni-kassel.de, marco.saelzer@rptu.de

Abstract

We investigate the computational complexity of neural network verification in quantised settings. We distinguish three classes of Feedforward Neural Networks (FNNs): rational FNNs with exact rational weights, quantised FNNs whose weights come from a finite-width arithmetic, and dynamically quantised FNNs in which rational networks are evaluated with respect to a given finite-width arithmetic. We consider two types of specifications used in the literature. Linear programming (LP) specifications are conjunctions of linear constraints, while bit-vector (BV) specifications allow reasoning at the bit level and can express non-linear constraints. Our results give a complexity landscape of these verification problems. For quantised FNNs with fixed arithmetic precision, we show that verification under both LP and BV specifications remains NP-complete, matching the complexity of the rational case. For dynamically quantised FNNs with BV specifications, we establish upper bounds, complementing a previously known PSPACE-hardness result.

1 Introduction

Feedforward Neural Networks (FNN) are ubiquitous in deep learning, not only as a stand-alone model for classifying inputs or transforming input values into outputs, but also as components in other machine learning models, for instance transformers, cf. [Vaswani *et al.*, 2017]. The ever-growing quest for machine-learning methods to solve various computational tasks, in particular for safety-critical applications, comes with a need for methods that formally verify properties like safety or robustness of such models, including FNN.

In this paper, we are concerned with verification tasks of the following form: given an FNN N and a safety property φ, decide whether φ holds for N or not. To distinguish this from other forms of verification, such as testing N on a sample set of inputs or adjusting the training procedure to meet certain safety criteria, we refer to this as *static (FNN) verification*.

In its most rigorous form, static verification tasks are stated as formal decision problems. In one of the most prominent variants, addressed by a wide-range of practical solvers, cf. [Brix *et al.*, 2024; Brix *et al.*, 2023], or analysed formally, cf. [Sälzer and Lange, 2022; Wurm, 2023; Henzinger *et al.*, 2021], static verification is concerned with the question of whether the pairs of inputs and corresponding outputs of a given FNN satisfy certain linear constraints. Problems of this kind are often referred to as *reachability* problems, cf. [Huang *et al.*, 2020], and other formal problems like robustness can usually be reduced to reachability.

The reachability problem for FNN was claimed (see [Katz *et al.*, 2017; Katz *et al.*, 2022; Ruan *et al.*, 2018]) and shown (see [Sälzer and Lange, 2021; Sälzer and Lange, 2022]) to be NP-complete. In this setting, FNN are considered to be mappings on real-valued vector spaces. In practice, the arithmetic operations internal to the FNN are of course executed not in arithmetic over $\mathbb{R}$ which would be infeasible on any bounded-memory computer, but in some fixed-width arithmetic using 64bit floating-point or fixed-point numbers for instance. This shift from real-valued inputs and outputs to such discrete values is also known as *quantisation*, and it comes not only with a loss of precision but also with artefacts like rounding or over-/underflows. One may therefore argue that (I) NP-completeness of the reachability problem for FNN in arithmetic over $\mathbb{R}$ does not necessarily predict the computational complexity of formal verification for FNN over a quantised arithmetic, and (II) that the quantised setting is more interesting for practical purposes anyway.

We investigate the computational complexity of formal verification, i.e. reachability, for quantised FNN. Quantisation does not make formal verification easier, i.e. NP-hardness remains a lower bound. This is not really surprising since computational hardness is often tightly linked to discretisation, see for instance the NP-completeness of Mixed-Integer Linear Programming versus the polynomial-time solvability of $\mathbb{R}$-valued linear programs. [Henzinger *et al.*, 2021] recently studied the verification problem for quantised FNN and provided a result that suggests higher complexity, namely PSPACE instead of NP. There seems to be some widespread misunderstanding about the actual results, perhaps fuelled by the fact that the paper is seen to be hard to read. First of all, [Henzinger *et al.*, 2021] only provides a lower bound of PSPACE-hardness, not PSPACE-completeness. Second, they consider a richer specification language for safety properties: while NP-completeness holds for safety properties that are convex sets of vectors, i.e. spec-

ifications in the form of linear programs, [Henzinger *et al.*, 2021] uses bit-vector logics which allow non-convex sets to be specified. Thus, the increase in complexity from NP to PSPACE is not due to quantisation, as that paper may suggest, but due to the use of a richer specification language for safety properties.

In this paper, we aim to provide an overall picture of the computational complexity of formal verification for quantised FNN. We classify the complexities depending on (I) the involved specification formalisms for safety properties, as they are commonly used in the literature, (II) the exact nature of the fixed bit-width arithmetic that determines how the FNN's internal computations are performed on common hardware architectures and, finally, (III) the dependence on the parameter determining the bit-width of such numbers, in particular whether this is given in unary or in binary. The motivation for studying the effect that this parameter has, also comes from practice: suppose we import a trained FNN to run on our hardware. The FNN may come with advice saying that arithmetic operations should be carried out with a certain bit-width. In this case, that width would normally be given as a decimal number, i.e. the actual width is exponential in the representation size of that parameter. On the other hand, when verifying a quantised FNN on the hardware where it already resides, the bit-width rather corresponds to the size of actual CPU registers, and therefore a unary encoding of this length may be a more realistic measure of the space needed to represent such quantised values.

The paper is organised as follows. In Section 2, we provide thorough definitions of all fundamentals necessary for the findings presented in this paper. In Section 3, we formally present reachability problems of FNN as decision problems. In Section 4, we examine the complexity of reachability in different quantisation settings where specifications are given by linear constraints. In Section 5, we study the complexity of verifying quantised FNN in the context of bit-vector logic, complementing the work of [Henzinger *et al.*, 2021]. Finally, in Section 6, we briefly discuss our findings and outline prospective next steps.

2 Preliminaries

Numbers, vectors and matrices. We exclusively consider numbers from $\mathbb{Q}$. Typically, we denote vectors using small, bold symbols like $\boldsymbol{x}$, $\boldsymbol{y}$ or $\boldsymbol{z}$ and matrices using big, bold symbols like $\boldsymbol{A}$, $\boldsymbol{B}$ or $\boldsymbol{C}$. We denote by $\boldsymbol{x}_i$ the i-th entry of vector $\boldsymbol{x}$ and by $\boldsymbol{A}_{i,j}$ the entry in the i-th row and j-th column of matrix $\boldsymbol{A}$. Let $x \in \mathbb{Q}$ with $p \in \mathbb{Z}$ and $q \in \mathbb{N}^{>0}$ such that $x = \frac{p}{q}$. If not stated otherwise, we define the *size of* x, denoted by $|x|$, as $|x| = |p|+|q|$ where $|p| = (\lfloor \log_2(p)+1 \rfloor)+1$ and $|q| = \lfloor \log_2(q)+1 \rfloor$. Analogously, we extend the definition of $|\cdot|$ to vectors $\boldsymbol{x} \in \mathbb{Q}^m$ and matrices $\boldsymbol{A} \in \mathbb{Q}^{m \times n}$ by $|\boldsymbol{x}| = \sum_{i=1}^{m} |\boldsymbol{x}_i|$ and $|\boldsymbol{A}| = \sum_{i=1}^{m}\sum_{j=1}^{n} |\boldsymbol{A}_{i,j}|$.

Finite-width arithmetics. We consider two kinds of finite arithmetics: fixed- and floating-point arithmetics. A *fixed-point arithmetic* $\mathbb{F}^{\text{ix}}_{b,f}$ is a tuple $(\mathbb{Q}^{\text{fix}}_{b,f}, o, r)$, where $\mathbb{Q}_{b,f} = \{\frac{n}{2^f} \mid -2^{b-1} \leq n < 2^{b-1}, n \in \mathbb{Z}\}$ is the sets of rationals representable using b bits with f bits for representing the fractional parts, $o\colon \mathbb{Q} \to \mathbb{Q}_{b,f}$ is a map called *overflow mode* and $r\colon \mathbb{Q} \to \mathbb{Q}_{b,f}$ is a map called *rounding mode*. We associate with $\mathbb{F}^{\text{ix}}_{b,f}$ the typical arithmetic operations, carried out as $\circ_{\mathbb{F}^{\text{ix}}_{b,f}} := o(r(x \circ y))$ for all $\circ \in \{+, \cdot, \div\}, x, y \in \mathbb{Q}$, and binary comparisons, carried out as $\sim_{\mathbb{F}^{\text{ix}}_{b,f}} := o(x) \sim o(y)$ for all $\sim \in \{<, \leq, =\}, x, y \in \mathbb{Q}$. Similarly, a *floating-point arithmetic* $\mathbb{F}^{\text{float}}_{p,e}$ is a tuple $(\mathbb{Q}^{\text{float}}_{p,e}, r)$ where

$$\mathbb{Q}^{\text{float}}_{p,e} = \{(-1)^s \cdot (1 + \frac{k}{2^p}) \cdot 2^E \mid s \in \{0,1\}, E, k \in \mathbb{N}, \\ -(2^{e-1}-2) \leq E \leq 2^{e-1}-1, 0 \leq k \leq 2^p - 1\}$$

is the set of rationals representable with p-bit significant and e-bit exponent width, and $r\colon \mathbb{Q} \to \mathbb{Q}^{\text{float}}_{p,e}$ is a rounding mode. We associate with $\mathbb{F}^{\text{float}}_{p,e}$ the typical arithmetic operations, carried out as $\circ_{\mathbb{F}^{\text{float}}_{p,e}} := r(x \circ y)$ for all $\circ \in \{+, \cdot, \div\}, x, y \in \mathbb{Q}$, and binary comparisons, carried out as $\sim_{\mathbb{F}^{\text{float}}_{p,e}} := r(x) \sim r(y)$ for all $\sim \in \{<, \leq, =\}$ and $x, y \in \mathbb{Q}$. To keep things concise, we refer to both of these as *finite-width arithmetics* $\mathbb{F}_{n_1,n_2}$ and represent them as tuples $\mathbb{F} = (n_1, n_2, r, o)$, where n_1, n_2 are the total bit number and fractional width (fixed-point), or the significant and exponent width (floating-point); r is the rounding mode and o the overflow mode (in the case of floating-point, we simply assume $o = r$ w.l.o.g.). We use $\mathbb{F}_{n_1,n_2}$ to refer to the full arithmetic as well as the set of numbers representable in $\mathbb{F}_{n_1,n_2}$.

Feedforward neural networks. An FNN N, of some class of FNN $\mathcal{N}$, is a tuple $(\boldsymbol{A}^{(i)}, \boldsymbol{b}^{(i)})_{i=1}^{k}$ of matrices $\boldsymbol{A}^{(i)} \in \mathbb{Q}^{m_i \times n_i}$ and vectors $\boldsymbol{b}^{(i)} \in \mathbb{Q}^{m_i}$. We call k the *number of layers (or depth) of* N, n_1 the *input dimension of* N, n_k the *output dimension of* N, m_i the *size of layer* i for all $i \leq k$, and generally assume that $m_i = n_{i+1}$ for all $i \leq k-1$. This concise representation of FNN from $\mathcal{N}$ assumes unambiguity in the definition of $\mathcal{N}$ regarding which and how activation functions are used by $N \in \mathcal{N}$. We generally assume that this is $\text{relu}(x) = \max(0, x)$, the prominent *ReLU activation* function, across all layers.[1]

Regarding different assumptions about the underlying arithmetic, we consider two distinct types of FNN for verification:

Rational FNN: A *rational FNN* $N = (\boldsymbol{A}^{(i)}, \boldsymbol{b}^{(i)})_{i=1}^{k}$ has weights and biases from $\mathbb{Q}$. It computes a function $N\colon \mathbb{Q}^{n_1} \to \mathbb{Q}^{n_k}$ by the standard forward propagation with $N(\boldsymbol{x}) =$

$$\sigma^{(k)}(\boldsymbol{A}^{(k)} \cdot (\sigma^{(k-1)}(\cdots \sigma^{(1)}(\boldsymbol{A}^{(1)} \cdot \boldsymbol{x} + \boldsymbol{b}^{(1)}) \cdots)) + \boldsymbol{b}^{(k)}) \,.$$

For rational FNNs, we define $|N| = \sum_{i=1}^{k} |\boldsymbol{A}^{(i)}| + |\boldsymbol{b}^{(i)}|$.

Quantised FNN: Given some finite-width arithmetic $\mathbb{F} = (n_1, n_2, r, o)$, a *quantised FNN* has weights and biases from $\mathbb{F}_{n_1,n_2}$. Such an FNN $N|_{\mathbb{F}}$ computes a function $N|_{\mathbb{F}}\colon \mathbb{F}^{n_1} \to \mathbb{F}^{n_k}$ by $N|_{\mathbb{F}}(\boldsymbol{x}) =$

$$\sigma^{(k)}_{\mathbb{F}}(\boldsymbol{A}^{(k)} \cdot_{\mathbb{F}} (\sigma^{(k-1)}_{\mathbb{F}}(\cdots \sigma^{(1)}_{\mathbb{F}}(\boldsymbol{A}^{(1)} \cdot_{\mathbb{F}} \boldsymbol{x} +_{\mathbb{F}} \boldsymbol{b}^{(1)}) \cdots)) +_{\mathbb{F}} \boldsymbol{b}^{(k)})$$

[1] As done in other works such as [Sälzer and Lange, 2022], our results can be extended to FNN using general *piecewise linear* activation functions.

where $\sigma^{(i)}(\cdot)$ denotes the entrywise application of activation function $\sigma^{(i)}\colon \mathbb{Q} \to \mathbb{Q}$. We assume that the definition of $\mathbb{F}$ determines how $\sigma^{(i)}$ is carried out in $\mathbb{F}$. We associate with $N|_{\mathbb{F}}$ the tuple $(\boldsymbol{A}_{\mathbb{F}}^{(i)}, \boldsymbol{b}_{\mathbb{F}}^{(i)})_{i=1}^{k}$ where all weights and biases are represented in $\mathbb{F}_{n_1,n_2}$. For quantised FNN, we define $|N|_{\mathbb{F}}| = \sum_{i=1}^{k} |\boldsymbol{A}^{(i)}|_{\mathbb{F}} + |\boldsymbol{b}^{(i)}|_{\mathbb{F}}$, where $|\cdot|_{\mathbb{F}}$ denotes the representation size as specified by $\mathbb{F}$.

Linear Programs. A *linear program (LP)* is a system of linear constraints of the form $L = \bigwedge_{i=1}^{m} \left(\sum_{j=1}^{n} a_{i,j} \cdot x_j \sim_i b_i \right)$ where $a_{i,j}, b_i \in \mathbb{Q}$, x_j are variables, and $\sim_i \in \{<, \leq, =, \geq, >\}$ for each i. The *feasibility problem* for such linear programs is: given an LP, does there exist an assignment of values to the variables $x_1, \ldots, x_n$, that satisfies all constraints? We can encode neural network verification problems as feasibility problems over linear constraints, where the constraints capture both the network's behaviour and the property specification. We define $|L| = \sum_{i=1}^{m} |b_i| + \sum_{j=1}^{n} |a_{i,j}|$.

A fundamental result in linear programming theory establishes a polynomial bound on the size of solutions:

Theorem 1 ([Korte and Vygen, 2018])**.** *Let L be a linear program with input size $|L|$.If L has a solution, then there exists a solution $(x_1, \ldots, x_n)$ such that $\sum_{j=1}^{n} |x_j| \leq poly(|L|)$.*

This polynomial bound is crucial for our analysis as it ensures that solutions to linear programs used in neural network verification have representations whose size is polynomially bounded by the input representation size.

Bit-Vector Logics (BV). We consider fixed-size bit vector logics as introduced in [Kovásznai *et al.*, 2016]. Let $\ell \in \mathbb{N}^+$ be a fixed bit-width and $\mathcal{V}$ be a finite set of variables. A BV formula φ is inductively defined as

$$\varphi := (t_1 = t_2) \mid \neg\varphi \mid \varphi \wedge \varphi \quad t := x \mid c \mid \sim t \mid t \,\&\, t \mid (t \mid t)$$

where $x \in \mathcal{V}$ and $0 \leq c \leq 2^{\ell} - 1$.

A BV formula φ is evaluated over bit-vectors of fixed size ℓ. An assignment is a function $\vartheta : \mathcal{V} \to \{0,1\}^{\ell}$ that maps each variable to a bit-vector of length ℓ. The semantics of formulas is defined inductively as

$$\begin{aligned} \vartheta \models (t_1 = t_2) &\Leftrightarrow [\![t_1]\!]^{\vartheta} = [\![t_2]\!]^{\vartheta} \\ \vartheta \models \varphi_1 \wedge \varphi_2 &\Leftrightarrow \vartheta \models \varphi_1 \text{ and } \vartheta \models \varphi_2 \\ \vartheta \models \neg\varphi &\Leftrightarrow \vartheta \not\models \varphi \end{aligned}$$

The interpretation of terms under an assignment ϑ is defined as

$$\begin{aligned} [\![x]\!]^{\vartheta} &= \vartheta(x) & [\![c]\!]^{\vartheta} &= \mathrm{enc}_{\ell}(c) \\ [\![\sim t]\!]^{\vartheta} &= \sim [\![t]\!]^{\vartheta} & [\![t_1 \odot t_2]\!]^{\vartheta} &= [\![t_1]\!]^{\vartheta} \odot [\![t_2]\!]^{\vartheta} \end{aligned}$$

for $\odot \in \{\&, |\}$, where '&', '|' and '$\sim$' denote the standard bitwise AND, OR, and NOT operations and $\mathrm{enc}_{\ell}(c)$ is the ℓ-bit binary representation of c.

A BV formula φ is *satisfiable over bit-vectors of length ℓ* if there exists an assignment ϑ such that $\vartheta \models \varphi$.

Proposition 2 ([Kovásznai *et al.*, 2016])**.** *The satisfiability problem for BV formulas with bit length and constants given in binary is* NP*-complete.*

The *model checking* problem for BV is: given a BV formula φ with bit-width b and an assignment ϑ, determine whether $\vartheta \models \varphi$.

Proposition 3. *The model-checking problem for BV formulas can be decided in polynomial time.*

Proof. This is trivial, as given an assignment ϑ, one can evaluate φ bottom-up by traversing the syntax tree. Each term can be evaluated in time $\mathcal{O}(b)$, thus the whole procedure can be done in time $\mathcal{O}(b \cdot |\varphi|)$. □

3 Verifying Quantised FNN: Overview

We consider five distinct reachability problems in this paper, differentiated by the type of arithmetic presumed for numerical operations and two separate formalisms utilised to specify valid inputs and outputs in the literature. We start with the most standard, yet somewhat idealised setting.

REACH$_{\mathbb{Q}}$(LP): given rational FNN N, and two rational linear programs L_1, L_2, decide whether there is input $\boldsymbol{x}$ of N that satisfies L_1 such that the output $N(\boldsymbol{x})$ satisfies L_2.

REACH$_{\mathbb{Q}}$(LP) assumes the computations of N, including the representation of its numerical parameters, to be performed in full-precision, rational arithmetic. However, this assumption is unrealistic for concrete computer architectures, which are limited to specific finite-width arithmetics, such as IEEE 754 floating-point arithmetic using 32 bits to represent numbers in binary. Neglecting these real-world restrictions, as done by various solvers addressing REACH$_{\mathbb{Q}}$(LP) or similar problems relying on idealised assumptions, has severe consequences as shown in [Jia and Rinard, 2021]. Thus, it is crucial to account for the quantisation of FNN due to some finite-width arithmetic during verification. Let $\mathcal{F} = \{\mathbb{F}_{n_1,n_2} \mid n_1, n_2 \in \mathbb{N}\}$ denote the set of all finite-width arithmetics as defined in Section 2. We assume w.l.o.g. that $\mathcal{F}$ only contains well-defined arithmetics, excluding, for instance, fixed-point arithmetics $\mathbb{F}_{b,f}^{\mathrm{ix}}$ with $f > b$.

REACH$_{\mathcal{F}}$(LP): given a quantised FNN $N|_{\mathbb{F}}$ with input dimensionality d, and two quantised linear programs $L_1|_{\mathbb{F}}, L_2|_{\mathbb{F}}$ for some $\mathbb{F} \in \mathcal{F}$, decide whether there is $\boldsymbol{x} \in \mathbb{F}^d$ that satisfies $L_1|_{\mathbb{F}}$ such that $N|_{\mathbb{F}}(\boldsymbol{x})$ satisfies $L_2|_{\mathbb{F}}$.

In the context of quantised FNN, [Henzinger *et al.*, 2021] considered specifications other than plain linear constraints, namely those expressed by bit-vector logic (BV); see Section 2 for details.

REACH$_{\mathcal{F}}$(BV): given a quantised FNN $N|_{\mathbb{F}}$ with input dimensionality d for some $\mathbb{F} = \mathbb{F}_{n_1,n_2} \in \mathcal{F}$, and two BV formulas φ_1, φ_2 both with bit-length $\ell = n_1 + n_2$, decide whether there is $\boldsymbol{x} \in \mathbb{F}^d$ that satisfies φ_1 such that $N|_{\mathbb{F}}(\bar{x})$ satisfies φ_2.

While REACH$_{\mathcal{F}}$(LP) and REACH$_{\mathcal{F}}$(BV) realistically assume restrictions imposed by finite-width arithmetic, they permit further generalisation: quantisation often occurs post-training, cf. [Gholami *et al.*, 2021], implying that the FNN is initially trained and delivered using more exacting representations. In actual implementations, it is quantised to different small-width arithmetics, meaning that the size of the FNN and the size of the finite-width arithmetic are independent.

REACH$(\mathcal{F}, \mathbf{LP})$: given a finite-width arithmetic $\mathbb{F} \in \mathcal{F}$, a rational FNN N with input dimensionality d, and two rational linear programs L_1, L_2, decide whether there exists $\boldsymbol{x} \in \mathbb{F}^d$ that satisfies $L_1|_{\mathbb{F}}$ such that $N|_{\mathbb{F}}(\boldsymbol{x})$ satisfies $L_2|_{\mathbb{F}}$.

REACH$(\mathcal{F}, \mathbf{BV})$: given a finite-width arithmetic $\mathbb{F} \in \mathcal{F}$, a rational FNN N with input dimensionality d, and two BV formulas φ_1, φ_2 with bit-length $\ell = n_1 + n_2$, decide whether there exists $\boldsymbol{x} \in \mathbb{F}^d$ that satisfies φ_1 such that $N|_{\mathbb{F}}(\boldsymbol{x})$ satisfies φ_2.

We remark that post-training quantisation techniques, as assumed in REACH$(\mathcal{F}, \text{LP})$ and REACH$(\mathcal{F}, \text{BV})$, typically involve more than merely rounding numerical parameters using standard rounding methods specified by some finite-width arithmetic $\mathbb{F}$, like rescaling weights prior to quantisation. Nonetheless, we note that such variations do not affect the aforementioned problems because they examine the safety of the resulting quantised FNN, independent of its relation to the unquantised version.

4 Verifying Quantised FNN in Relation to Linear Constraints

REACH$_{\mathbb{Q}}$(LP) is the problem considered most often in the context of reachability w.r.t. linear constraints. It assumes FNN to work over full-precision, rational arithmetic and input- and output-specifications given by sets of linear constraints, i.e. linear programs.

Theorem 4 ([Sälzer and Lange, 2021])**.** REACH$_{\mathbb{Q}}$*(LP) is* NP*-complete.*

We briefly recapitulate the proof of Theorem 4 to reuse certain arguments for subsequent results. NP-membership is shown using the certificate-based understanding of NP: guess a witness w of polynomial size in $|N| + |L_1| + |L_2|$ which is the size of an instance (N, L_1, L_2) of REACH$_{\mathbb{Q}}$(LP), and then verify in polynomial time relative to $|w| + |N| + |L_1| + |L_2|$ whether (N, L_1, L_2) is a valid instance. Here, the key insight is that for each input $\boldsymbol{x}$ of N, there is a fixed *activation pattern* of N, meaning a tuple $w \in \{0,1\}^n$, where n is the number of nodes in N, such that $w[i] = 1$ if and only if the input of the ith node in N is greater than or equal to 0. Clearly, such an activation pattern w is of size polynomial in the size of N. Having this, the argument for NP-membership is as follows: given FNN N, LP L_1 and L_2, (I) guess an activation pattern w of N. (II) Fix all ReLU nodes of N by replacing the activation of the ith node by the identity if $w[i] = 1$, and by the constant 0-function if $w[i] = 0$. (III) Construct a single LP L by combining L_1, L_2, and N, which is possible since N computes a linear function now, and add constraints ensuring that any solution respects w, and (IV) solve L. Since L is essentially just the combination of N, L_1, and L_2, and LPs can be solved in polynomial time, the procedure above runs in nondeterministic polynomial time.

NP-hardness of REACH$_{\mathbb{Q}}$(LP) is shown via a reduction from 3SAT, the satisfiability problem of propositional Boolean formulae in three-conjunctive normal form (3CNF). Given a 3CNF formula $\varphi = \bigwedge_{i=1}^{n} C_i$ with $n \in \mathbb{N}$ where each $C_i = (\ell_{i,1} \vee \ell_{i,2} \vee \ell_{i,3})$ is a clause with literals $\ell_{i,j}$ of the form X or $\neg X$ for propositional variables $X \in \mathcal{X}$, we construct an FNN N_φ that accepts precisely those inputs $\boldsymbol{x}$ that encode a satisfying assignment $I : \mathcal{X} \to \{0,1\}$ for the propositional variables used in φ. The construction of N_φ involves combining specific gadgets as presented in Fig. 1. We have one gadget for each kind of subformula in φ, mimicking their semantics in a straightforward manner, and combine them as determined by the structure of φ. To ensure that N_φ only accepts inputs corresponding to an encoded variable assignment, which is essential for a correct reduction from 3SAT, the FNN N_φ also includes an "$x \in \{0,1\}$" gadget (see Fig. 1) for each input dimension. Additionally, we use L_1 to ensure that each input is within $[0,1]$, which is necessary for the idea underlying the "$x \in \{0,1\}$"-gadgets, as well as L_2 to ensure that the output of the final "$\bigwedge_{i=1}^{n} C_i$" gadget in N_φ is exactly 1 and the output of all "$x \in \{0,1\}$" gadgets is exactly 0. For details, we refer to [Sälzer and Lange, 2021].

$$
\begin{aligned}
\varphi = \textstyle\bigwedge_{i=1}^{n} C_i &: \operatorname{relu}(\textstyle\sum_{i=1}^{n} x_{C_i} - (n-1)) \\
C_i = \textstyle\bigvee_{j=1}^{3} \ell_{i,j} &: \operatorname{relu}(\operatorname{relu}(\textstyle\sum_{j=1}^{3} x_{\ell_{i,j}}) - \operatorname{relu}(\textstyle\sum_{j=1}^{3} x_{\ell_{i,j}} - 1)) \\
\ell_{i,j} = X_{i,j} &: \operatorname{relu}(x_{i,j}) \\
\ell_{i,j} = \neg X_{i,j} &: \operatorname{relu}(1 - x_{i,j}) \\
x \in \{0,1\} &: \operatorname{relu}(\operatorname{relu}(\tfrac{1}{2} - x) + \operatorname{relu}(\tfrac{1}{2} - x) - \tfrac{1}{2})
\end{aligned}
$$

Figure 1: FNN gadgets, one for each component of a propositional 3CNF formula φ, that output 1 if the corresponding subformula is satisfied and 0 otherwise. Additionally, it shows a gadget that for $x \in [0;1]$ outputs 0 if $x = 0$ or $x = 1$, and some value greater than 0 otherwise.

Turning to quantised settings, we find that while specific arguments for Theorem 4 require additional support, the overall complexity of verifying reachability in the context of linear constraints remains unchanged. Remember that REACH$_{\mathcal{F}}$(LP) takes as input FNN and LP that are already quantised.

Theorem 5. REACH$_{\mathcal{F}}$*(LP) is* NP*-complete.*

Proof. Membership in NP relies on the certifier-based understanding of NP and is straightforward. Let $(N|_{\mathbb{F}}, L_1|_{\mathbb{F}}, L_2|_{\mathbb{F}})$ with $\mathbb{F} = \mathbb{F}_{n_1,n_2}$. (I) Guess an input $\boldsymbol{x} \in \mathbb{F}$, (II) check whether $L_1|_{\mathbb{F}}$ is satisfied, (III) compute $N|_{\mathbb{F}}(\boldsymbol{x})$, and (IV) check whether $N|_{\mathbb{F}}(\boldsymbol{x})$ satisfies $L_2|_{\mathbb{F}}$. Clearly, $|\boldsymbol{x}|$ is linear in $n_1 + n_2$, which is polynomial in $|N|$ since N contains numerical parameters represented in $\mathbb{F}$. Furthermore, steps (II) to (IV) can be done in polynomial time with respect to $|L_1| + |N| + |L_2|$ since all numerical parameters are represented in $\mathbb{F}$. Note that we assume constant time for basic arithmetic operations. The correctness is implied by Theorem 1, which states that feasible LP have solutions of polynomial size. The reasoning used for Theorem 4, as sketched above, carries over to parts (I)–(IV).

Regarding NP-hardness, we refer to the same argument used in Theorem 4, namely, the reduction from 3SAT translating a 3CNF formula φ into an FNN N_φ. The key insight is that the same reduction described above works here because the gadgets shown in Figure 1 use only low-precision parame-

ters, specifically 1, 0, and $\frac{1}{2}$, which are exactly representable in all non-trivial finite-width arithmetics $\mathbb{F}$. Moreover, the sums in the "$\bigwedge_{i=1}^{n} C_i$" and "$\bigvee_{j=1}^{3} \ell_{i,j}$" gadgets are at most n. Thus, it is sufficient to assume that $N_\varphi|_{\mathbb{F}}$ to be quantised by any $\mathbb{F}$ with at least $\lceil \log n \rceil$ bits to represent integer parts. □

The other quantised problem we consider in this section is $\text{REACH}(\mathcal{F}, \text{LP})$. In contrast to $\text{REACH}_{\mathcal{F}}(\text{LP})$, the given FNN contains full-precision rational parameters, and the quantisation to some finite-width arithmetic $\mathbb{F}_{n_1,n_2}$ is not assumed a priori, making $\mathbb{F}_{n_1,n_2}$ part of the input. At first glance, this may lead one to thinking that this problem must be more difficult, since we have to deal with numbers of bit size $n_1 + n_2$, and the parameters n_1, n_2 are given as binary numbers. However, it is easily argued that this is not the case.

Theorem 6. $\text{REACH}(\mathcal{F}, LP)$ *is* NP*-complete.*

Proof. NP-membership is shown as in Theorem 5: given $(\mathbb{F}_{n_1,n_2}, N, L_1, L_2)$, we quantise N, L_1, and L_2 using $\mathbb{F}_{n_1,n_2}$, guess an input $\boldsymbol{x} \in \mathbb{F}_{n_1,n_2}$ and check whether it is a witness for the validity of (N, L_1, L_2). Note that this assumes that the quantisation method used for N, L_1 and L_2 works in polynomial time. Given that, we need to argue why the size of $\boldsymbol{x}$ is polynomial in the size of $(\mathbb{F}_{n_1,n_2}, N, L_1, L_2)$. We make a case distinction. First, assume that $|n_1| + |n_2|$ is larger than the maximum size of any rational parameter of N, L_1 or L_2. Then, guessing $\boldsymbol{x}$, which is polynomial in $|n_1| + |n_2|$, is sufficient since this also subsumes solutions for the LP resulting from combining (N, L_1, L_2), utilising the activation pattern of N for $\boldsymbol{x}$, according to Theorem 1. Second, assume that $|n_1| + |n_2|$ is smaller than the maximum size of any rational parameter of N, L_1 or L_2. Then, guessing $\boldsymbol{x}$, which is polynomial in $|N| + |L_1| + |L_2|$ is also sufficient, again using Theorem 1. In combination, it is sufficient to take $\boldsymbol{x}$ which is polynomial in the size of $(\mathbb{F}_{n_1,n_2}, N, L_1, L_2)$.

NP-hardness is shown similarly to Theorems 4 and 5 by establishing a reduction from 3SAT. In this case, we introduce a finite-width arithmetic $\mathbb{F}_{n_1,n_2}$ with a sufficient number of bits. We observe that $n_1 + n_2$ is polynomial in the size of the 3CNF formula φ. □

5 Verifying Quantised FNN in Relation to BV Constraints

As seen in Section 4, using linear constraints always guarantees solutions of polynomial size with respect to the size of the FNN and specification LP. [Henzinger *et al.*, 2021] investigated the verification of quantised FNNs with respect to bit-vector logics, cf. [Kovásznai *et al.*, 2016], instead. This is motivated by the fact that BV allow non-linear constraints to be specified and single bits of the FNN's in- and output to be addressed. This can avoid rounding and overflow problems in contrast to the standard specifications via linear constraints, which crucially depend on the type of arithmetic and bit-width used when evaluating an FNN.

First, for already quantised FNN, these non-linear constraints do not change anything complexity-wise.

Theorem 7. $\text{REACH}_{\mathcal{F}}(BV)$ *is* NP*-complete.*

Proof. Membership in NP is shown as in Theorem 5. Let $(N|_{\mathbb{F}}, \varphi_1, \varphi_2)$ with $\mathbb{F} = \mathbb{F}_{n_1,n_2}$. (I) Guess an input $\boldsymbol{x} \in \mathbb{F}$, (II) check whether $\boldsymbol{x} \models \varphi_1$, (III) compute $N|_{\mathbb{F}}(\boldsymbol{x})$, and (IV) check whether $N|_{\mathbb{F}}(\boldsymbol{x}) \models \varphi_2$. First, $|\boldsymbol{x}|$ is linear in $n_1 + n_2$ which is polynomial in $|N|$ since N contains numerical parameters represented in $\mathbb{F}$. By Prop. 3, model checking for BV formulas can be done in polynomial time.

Regarding NP-hardness,we use the same argument as in Theorem 5. As the lower bound construction does not need any input or output specification, NP-hardness already holds for BV specifications as well. □

Analogously to Section 4, problem $\text{REACH}(\mathcal{F}, \text{BV})$ assumes the FNN given with full-precision parameters together with some finite-width arithmetic as part of the input. This problem was already considered in [Henzinger *et al.*, 2021], where only a lower bound was shown.

Theorem 8 ([Henzinger *et al.*, 2021])**.** $\text{REACH}(\mathcal{F}, BV)$ *is* PSPACE*-hard.*

The remainder of this section is dedicated to establishing upper bounds. While the lower bound in [Henzinger *et al.*, 2021] is independent of the arithmetic, the upper bounds depend on them. We therefore distinguish two cases and establish the following.

Theorem 9. $\text{REACH}(\mathcal{F}, BV)$ *is* PSPACE*-complete for fixed-point and floating-point arithmetic with constant exponent width.*

We show this by reducing $\text{REACH}(\mathcal{F}, \text{BV})$ to the problem of checking non-emptiness for non-deterministic finite automata (NFA), represented in a succinct way. Using automata to recognise relations induced by linear arithmetics (cf. [Wolper and Boigelot, 2000] and [Esparza and Blondin, 2023, Chp. 14]) or for recognising the input-output behaviour of rational FNN in particular [Sälzer *et al.*, 2024] was already explored in previous work. We show that these results can be extended to quantised FNNs as well and use it for a complexity analysis.

Unlike explicit automata defined by state transition tables, a *succinct NFA* describes a machine with an exponentially large state space via a compact, implicit representation. Formally, an NFA with state space of size 2^n is succinctly represented if each state has a polynomial description size in n and there are two polynomial-time algorithms: (1) TRANS takes two state descriptions q, q' and a symbol σ and decides whether there is a transition from q to q' with symbol σ. FINAL takes a state description q and decides whether q is a final state. The problem of deciding non-emptiness for such NFAs is equivalent to the reachability problem in succinctly represented graphs, which was explored in previous work.

Proposition 10 ([Galperin and Wigderson, 1983])**.** *The non-emptiness problem for succinctly represented NFA is in* PSPACE*.*

Succinctly represented NFAs are closed under intersection and union. The state of the product/union automaton is the tuple of the states of the component automata. The size of the combined state description is the sum of the individual sizes, which remains polynomial. Functions TRANS checks

for all components to be true. Function FINAL checks that both states are final (for intersection) or at least one is (for union).

The input-output behaviour of rational FNNs evaluated over a given fixed-width arithmetic can be captured by succinct NFA. We start with the case of fixed-point arithmetic, encoding fixed-point numbers over the alphabet $\Sigma = \{0, 1\}$ with least-significant bit first and in two-complement representation. I.e. the word $b_f b_{f-1} \cdots b_1 a_0 a_1 \cdots a_{b-1} \in \{0,1\}^{b+f}$ is the encoding of the number x with

$$x = \sum_{i=1}^{f} b_i \cdot 2^{-i} + \sum_{i=0}^{b-2} a_i \cdot 2^i + a_{b-1} \cdot -2^{b-1} .$$

Now fix an FNN N with parameters θ, input and output dimensions m and n, and a fixed-point arithmetic $\mathbb{F}^{\text{ix}}_{b,f}$ with bit-width $k = b + f$. Its input-output relation is $R_{N,\mathbb{F}^{\text{ix}}_{b,f}} = \{(x, y) \in \{0,1\}^{mk} \times \{0,1\}^{nk} \mid N|_{\mathbb{F}^{\text{ix}}_{b,f}}(x) = y\}$.

Lemma 11. *There exists a succinct NFA $M_{N,\mathbb{F}^{\text{ix}}_{b,f}}$ with description size in poly$(|N| + \log(b + f))$ accepting $R_{N,\mathbb{F}^{\text{ix}}_{b,f}}$.*

Proof. We construct $M_{N,\mathbb{F}^{\text{ix}}_{b,f}}$ to process the input-output relation bitwise. A state $q_t = (t, \boldsymbol{c}, \boldsymbol{r}, \boldsymbol{c}_{\text{init}}, \boldsymbol{w})$ is a tuple, where $t \in \{0, \cdots, k\}$ is a bit counter, $\boldsymbol{c} \in \mathbb{Z}^{|\theta|}$ represents a vector of carry values for all linear operations, $\boldsymbol{r} \in \mathbb{Q}^{|\theta|}$ is a fractional accumulator for rounding and $\boldsymbol{w} = \{0,1\}^{|\theta|}$ is the activation pattern. At $t = 0$, the automaton non-deterministically guesses the activation pattern $\boldsymbol{w}$ and the vector $\boldsymbol{c}_{\text{init}}$, which represents the carry bit that would be generated by rounding the fractional result of the scalar products. The accumulator $\boldsymbol{r}$ is initialised to 0. The accumulator aggregates computations shifted below the LSB, in order to verify the rounding decisions.

Note that for a single neuron with i inputs and maximum weight λ, the carry is bounded by $\mathcal{O}(i \cdot \lambda)$ and especially independent of the bit-width k. Additionally, the fractional accumulator is always bounded by the size of the weights, which are multiplied. Thus, each state needs space $\mathcal{O}(\log k + |N|)$ which is polynomial in the input size. Note that weights are not stored in the state. Instead, we use a function GetBit(w, j) that computes the j-th bit of a rational weight $w = p/q$ in polynomial time (see Lemma 15 in Appendix).

At step t, $M_{N,\mathbb{F}^{\text{ix}}_{b,f}}$ reads the t-th bits of input x and output y. It updates the carry state from $\boldsymbol{c}$ to $\boldsymbol{c}'$ by using the t-th bit of each weight, which can be computed in polynomial time (for detailed information on the calculation of carry values consider [Wolper and Boigelot, 2000]). Regarding the activation pattern, if $\boldsymbol{w}_i = 1$ then it checks whether y_i is the correct sum bit or $y_i = 0$, when $\boldsymbol{w}_i = 0$, meaning that the corresponding ReLU is inactive.

For $t = k$, a state is accepting when the corresponding sign bit of every neuron's internal sum is consistent with the activation pattern. Additionally, the automaton verifies consistency. For ReLU, if the accumulated carry indicates that the result should have been negative, but $\boldsymbol{w}_j = 1$, or positive but $\boldsymbol{w}_j = 0$, this may imply overflow or incorrect guessing. Specifically, if the carry into the sign bit differs from the carry out, an overflow occurred. The NFA checks that, if overflow to a positive value occurred, the output y represents the maximum representable value, and similarly for a negative overflow. Moreover, it is verified that the accumulated tail $\boldsymbol{r}$ justifies the initial guess $\boldsymbol{c}_{\text{init}}$. For the rounding mode round-to-nearest, it checks whether $\boldsymbol{r} \geq 0.5 \iff c_{\text{init}} = 1$. □

Next, we will consider the case of floating-point arithmetic. The crucial difference here is that while addition and multiplication in fixed-point arithmetic are local operations, meaning that the computation of the i-th bit depends only on the i-th bit of the operands and a carry from position $i - 1$, in floating-point arithmetic addition and multiplication involve an alignment step. When two numbers $x_1 = m_1 \cdot 2^{e_1}$ and $x_2 = m_2 \cdot 2^{e_2}$ have different exponents, the mantissa of the smaller number must be right-shifted by $\delta = |e_1 - e_2|$. Therefore, a finite automaton reading the numbers bitwise would need to buffer $\mathcal{O}(\delta)$ bits to align the operands. As k is given in binary, this would lead to a description of the resulting NFA of exponential size. It remains open whether there is another way to obtain a PSPACE upper bound for the general floating-point arithmetic or whether the lower bound of [Henzinger *et al.*, 2021] can be lifted. However, to obtain a PSPACE upper bound when using floating-point arithmetic, we can restrict to formats where the exponent width e is fixed, while the precision of the mantissa is variable. This is reasonable for hardware designs where dynamic range is static but precision varies (for example bf16 vs. fp32).

We encode floating-point numbers with bounded exponent width E and mantissa width k as a word $se_0 \cdots e_{E-1} m_1 \cdots m_k \in \{0,1\}^{E+k+1}$ encoding the number

$$x = (-1)^s \cdot 2^{\sum_{i=0}^{E-1} e_i \cdot 2^i} \cdot (1 + \sum_{i=1}^{k} m_i \cdot 2^{-i}) .$$

Lemma 12. *There exists a succinct NFA $M_{N,\mathbb{F}^{\text{loat}}_{k,E}}$ with description size in poly$(|N| + \log(k))$ accepting $R_{N,\mathbb{F}^{\text{loat}}{}_{k,E}}$.*

Proof. A state q_t in $M_{N,\mathbb{F}^{\text{loat}}_{k,E}}$ is a tuple $(t, \boldsymbol{E}, \boldsymbol{S}, \boldsymbol{c}, \boldsymbol{r}, \boldsymbol{c}_{\text{init}}, \boldsymbol{w})$, where $(t, \boldsymbol{c}, \boldsymbol{r}, \boldsymbol{c}_{\text{init}}, \boldsymbol{w})$ is as in the fixed-point case. Additionally, $\boldsymbol{E} \in \{0, \cdots, 2^E - 1\}^{|\theta|}$ stores the exponent values for every neuron and $\boldsymbol{S} \in (\{0,1\}^E)^{|\theta|}$ stores the last E mantissa bits of every neuron's activation value. Since E is fixed, $\boldsymbol{E}$ and $\boldsymbol{S}$ only take polynomial space, and each state needs space $\mathcal{O}(\log k + |N|)$, which is polynomial in the input size.

The construction relies on E being constant in three ways. *Alignment*: since all exponents lie in $\{0, \ldots, 2^E - 1\}$, the maximum alignment shift is $2^E - 1$, a constant; the mantissa bits that fall into this window are exactly those stored in $\boldsymbol{S}_j$, so aligned operands can be recovered from the state alone. *Normalisation*: after addition or multiplication of normalised values the leading bit shifts by a constant amount bounded by E, so normalisation is a constant-size state update. *Rounding*: the sub-LSB contributions are tracked by $\boldsymbol{r}$ as in the fixed-point case; the rounding decision depends only on $\boldsymbol{r}$ and the LSB, so the fixed-point rounding argument carries over directly. The t-th bit of every rational weight can be extracted

	Linear specifications	**Bit-vector specifications**
$\text{REACH}_{\mathbb{Q}}$	NP-complete ([Sälzer and Lange, 2021])	
$\text{REACH}_{\mathcal{F}}$	NP-complete (Thm. 5)	NP-complete (Thm. 7)
$\text{REACH}(\mathbb{F}^{\text{ix}}, \_)$	NP-complete (Thm. 6)	PSPACE-complete (Thm. 9, [Henzinger *et al.*, 2021])
$\text{REACH}(\mathbb{F}_E^{\text{loat}}, \_)$	NP-complete (Thm. 6)	PSPACE-complete (Thm. 9, [Henzinger *et al.*, 2021])
$\text{REACH}(\mathbb{F}^{\text{loat}}, \_)$	NP-complete (Thm. 6)	PSPACE-hard, $\in$ NEXPTIME (Prop. 14, [Henzinger *et al.*, 2021])

Table 1: Overview of the results established in this paper.

in polynomial time (see Lemma 16 in Appendix), so TRANS and FINAL both run in polynomial time. □

Lemma 13. *Let φ a BV formula over variables $x_1, \ldots, x_k$ and bit-width b (given in binary). There exists a succinct NFA M_φ with description size in poly$(|\varphi| + \log(b))$, such that M_φ accepts the set of models of φ.*

Proof. Assume φ to be in negation normal form, i.e. it consists of disjunction and conjunctions of atomic formulas of the form $(t_1 = t_2)$ or $\neg(t_1 = t_2)$. As succinct NFA are closed under union and intersection, it is sufficient to construct succinct NFA for each atomic subformula.

Let ψ be such a term. A state is a tuple $q = (j, f_{eq})$, where $j \in \{0, \ldots, b\}$ is a bit-counter and $f_{eq} \in \{0, 1\}$ is a flag tracking whether the evaluated terms equal up to position j. The description size is in $\mathcal{O}(b)$, which is polynomial. The start state is $(0, 1)$. Let $\sigma \in \{0, 1\}^{|V|}$ be the input symbol representing the j-th bit-slice of the variables. We define a recursive evaluation function $\text{Eval}(t, j, \sigma)$ for a term t: If $t = x \in V$ then $\text{Eval}(x, j, \sigma) = \sigma(x)$, if $t = c$ then $\text{Eval}(c, j, \sigma) = c_j$ (the j-th bit of c), if $t = (t_a \odot t_b)$ for $\odot \in \{\&, |, \oplus\}$ then $\text{Eval}(t, j, \sigma) = \text{Eval}(t_a, j, \sigma) \odot \text{Eval}(t_b, j, \sigma)$ and if $t = \sim t_a$ then $\text{Eval}(t, j, \sigma) = 1 - \text{Eval}(t_a, j, \sigma)$.

Since the depth of the term tree is bounded by $|\psi|$, Eval runs in polynomial time.

Algorithm $\text{TRANS}((j, f_{eq}), \sigma, (j', f'_{eq}))$ returns true iff $j' = j + 1$ and $f'_{eq} = 1$ if $f_{eq} = 1$ and $\text{Eval}(t_1, j, \sigma) = \text{Eval}(t_2, j, \sigma)$ or $f'_{eq} = 0$ otherwise. This transition logic ensures that f_{eq} starts at 1 and acts as a "sink": once a mismatch is detected at position j, f_{eq} becomes and remains 0 for all subsequent steps.

Algorithm $\text{FINAL}((j, f_{eq}))$ returns true iff $j = b$ and the flag matches the formula type: if ψ is $(t_1 = t_2)$ then accept iff $f_{eq} = 1$ (all bits matched). If ψ is $\neg(t_1 = t_2)$ then accept iff $f_{eq} = 0$ (at least one bit mismatch occurred). □

We are now ready to prove Theorem 9.

Proof (of Thm. 9). We aim to check whether there is an $\boldsymbol{x}$ such that $(\boldsymbol{x}, N|_{\mathbb{F}}) \models \varphi_1 \wedge \varphi_2$. This corresponds to the non-emptiness of the intersection of the FNN automaton and the specification automaton for $L(M) = L(M_{N,\mathbb{F}}) \cap L(M_{\varphi_1 \wedge \varphi_2})$. Succinct NFA are closed under intersection. We can therefore construct the succinct product NFA and check for emptiness using the PSPACE algorithm from Proposition 10. PSPACE-hardness follows from Theorem 8. □

For general floating-point arithmetic, we can derive an upper bound from Theorem 7.

Proposition 14. $\text{REACH}(\mathcal{F}, BV)$ *is in* NEXPTIME *for general floating-point arithmetic.*

Proof. This is a direct consequence of Theorem 7. Due to the binary encoding of the bit-width b we can guess a witness of exponential length. Analogously to Theorem 7, we can check the witness in polynomial time. □

These results complement the recently established results from [Henzinger *et al.*, 2021] with upper bounds.

6 Outlook

We established the complexity landscape of verifying quantised FNN (see Table 1 for an overview), showing that the complexity in most cases matches the complexity of verification in full rational arithmetic. Current NN verification tools operate under idealised real-valued assumptions, while actual implementations use finite-width arithmetic. As shown in [Jia and Rinard, 2021], this mismatch can invalidate formal guarantees. Our results show that, for the standard case of linear specifications, verification of quantised networks remains NP. Hence, developing sound verification tools for quantised FNNs is not only necessary for correctness, but feasible without any asymptotic overhead compared to existing real-valued solvers. Moreover, we complemented the previous PSPACE lower bound for verifying dynamically quantised FNN with bit-vector specifications by giving matching upper bounds in the cases of fixed-point and floating-point arithmetic with bounded exponent width. We identified that in case of general floating-point arithmetic it seems like the problem could be even harder than PSPACE, due to the fact that the buffer size needed for alignment prevents the PSPACE upper bound. It remains open whether this need for alignment can be exploited to show a matching NEXPTIME lower bound for this case. Additionally, investigating the practical implementation of our succinct automata approach could offer a promising direction for creating solvers that accurately verify networks in quantised settings.

References

[Brix *et al.*, 2023] Christopher Brix, Mark Niklas Müller, Stanley Bak, Taylor T. Johnson, and Changliu Liu. First three years of the international verification of neural networks competition (VNN-COMP). *Int. J. Softw. Tools Technol. Transf.*, 25(3):329–339, 2023.

[Brix *et al.*, 2024] Christopher Brix, Stanley Bak, Taylor T. Johnson, and Haoze Wu. The fifth international verification of neural networks competition (VNN-COMP 2024): Summary and results. *CoRR*, abs/2412.19985, 2024.

[Esparza and Blondin, 2023] Javier Esparza and Michael Blondin. *Automata Theory: An Algorithmic Approach*. MIT Press, Cambridge, MA, USA, October 2023.

[Galperin and Wigderson, 1983] Hana Galperin and Avi Wigderson. Succinct representations of graphs. *Inf. Control.*, 56(3):183–198, 1983.

[Gholami *et al.*, 2021] Amir Gholami, Sehoon Kim, Zhen Dong, Zhewei Yao, Michael W. Mahoney, and Kurt Keutzer. A survey of quantization methods for efficient neural network inference. *CoRR*, abs/2103.13630, 2021.

[Henzinger *et al.*, 2021] Thomas A. Henzinger, Mathias Lechner, and Đorđe Žikelić. Scalable verification of quantized neural networks. *Proceedings of the AAAI Conference on Artificial Intelligence*, 35(5):3787–3795, 2021.

[Huang *et al.*, 2020] Xiaowei Huang, Daniel Kroening, Wenjie Ruan, James Sharp, Youcheng Sun, Emese Thamo, Min Wu, and Xinping Yi. A survey of safety and trustworthiness of deep neural networks: Verification, testing, adversarial attack and defence, and interpretability. *Comput. Sci. Rev.*, 37:100270, 2020.

[Jia and Rinard, 2021] Kai Jia and Martin C. Rinard. Exploiting verified neural networks via floating point numerical error. In Cezara Dragoi, Suvam Mukherjee, and Kedar S. Namjoshi, editors, *Static Analysis - 28th International Symposium, SAS 2021, Chicago, IL, USA, October 17-19, 2021, Proceedings*, volume 12913 of *Lecture Notes in Computer Science*, pages 191–205. Springer, 2021.

[Katz *et al.*, 2017] Guy Katz, Clark W. Barrett, David L. Dill, Kyle Julian, and Mykel J. Kochenderfer. Reluplex: An efficient SMT solver for verifying deep neural networks. In Rupak Majumdar and Viktor Kuncak, editors, *Computer Aided Verification - 29th International Conference, CAV 2017, Heidelberg, Germany, July 24-28, 2017, Proceedings, Part I*, volume 10426 of *Lecture Notes in Computer Science*, pages 97–117. Springer, 2017.

[Katz *et al.*, 2022] Guy Katz, Clark W. Barrett, David L. Dill, Kyle Julian, and Mykel J. Kochenderfer. Reluplex: a calculus for reasoning about deep neural networks. *Formal Methods Syst. Des.*, 60(1):87–116, 2022.

[Korte and Vygen, 2018] Bernhard Korte and Jens Vygen. *Combinatorial Optimization: Theory and Algorithms*. Number 21 in Algorithms and Combinatorics. Springer Berlin Heidelberg, Berlin, Heidelberg, 6. 6th ed. 2018 edition, 2018.

[Kovásznai *et al.*, 2016] Gergely Kovásznai, Andreas Fröhlich, and Armin Biere. Complexity of fixed-size bit-vector logics. *Theory Comput. Syst.*, 59(2):323–376, 2016.

[Ruan *et al.*, 2018] Wenjie Ruan, Xiaowei Huang, and Marta Kwiatkowska. Reachability analysis of deep neural networks with provable guarantees. In Jérôme Lang, editor, *Proceedings of the Twenty-Seventh International Joint Conference on Artificial Intelligence, IJCAI 2018, July 13-19, 2018, Stockholm, Sweden*, pages 2651–2659. ijcai.org, 2018.

[Sälzer and Lange, 2021] Marco Sälzer and Martin Lange. Reachability is NP-complete even for the simplest neural networks. In *Reachability Problems: 15th International Conference, RP 2021, Liverpool, UK, October 25–27, 2021, Proceedings*, pages 149–164, Berlin, Heidelberg, 2021. Springer-Verlag.

[Sälzer and Lange, 2022] Marco Sälzer and Martin Lange. Reachability in simple neural networks. *Fundam. Informaticae*, 189(3-4):241–259, 2022.

[Sälzer *et al.*, 2024] Marco Sälzer, Eric Alsmann, Florian Bruse, and Martin Lange. Verifying and interpreting neural networks using finite automata. In Joel D. Day and Florin Manea, editors, *Developments in Language Theory*, pages 266–281, Cham, 2024. Springer Nature Switzerland.

[Vaswani *et al.*, 2017] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need. In Isabelle Guyon, Ulrike von Luxburg, Samy Bengio, Hanna M. Wallach, Rob Fergus, S. V. N. Vishwanathan, and Roman Garnett, editors, *Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems 2017, December 4-9, 2017, Long Beach, CA, USA*, pages 5998–6008, 2017.

[Wolper and Boigelot, 2000] Pierre Wolper and Bernard Boigelot. On the construction of automata from linear arithmetic constraints. In Susanne Graf and Michael I. Schwartzbach, editors, *Tools and Algorithms for Construction and Analysis of Systems, 6th International Conference, TACAS 2000, Held as Part of the European Joint Conferences on the Theory and Practice of Software, ETAPS 2000, Berlin, Germany, March 25 - April 2, 2000, Proceedings*, volume 1785 of *Lecture Notes in Computer Science*, pages 1–19. Springer, 2000.

[Wurm, 2023] Adrian Wurm. Complexity of reachability problems in neural networks. In Olivier Bournez, Enrico Formenti, and Igor Potapov, editors, *Reachability Problems - 17th International Conference, RP 2023, Nice, France, October 11-13, 2023, Proceedings*, volume 14235 of *Lecture Notes in Computer Science*, pages 15–27. Springer, 2023.