Appears in: Proceedings of the 34th International Conference on Real-Time Networks and Systems (RTNS 2026)

From Robustness to Worst-Case Timing Analysis: Reusing Neural Network Abstract Interpretation with DeepWCET

Tobias Häberlein[1], Dominik Penk[2], Dominik Riedelbauch[2], Isabella Stilkerich[2], and Peter Wägemann[1,3]

[1] Friedrich-Alexander-Universität Erlangen-Nürnberg (FAU), Erlangen, Germany
[2] Schaeffler Technologies AG & Co. KG, Herzogenaurach, Germany
[3] Karlsruhe Institute of Technology (KIT), Karlsruhe, Germany

tobias.haeberlein@fau.de

Abstract. Neural networks are increasingly deployed in safety-critical embedded systems, where every task needs a guaranteed worst-case execution time. For neural network inference, deriving such a bound is relatively straightforward because the computation is fixed by the network architecture. The main challenge lies in the code executed after inference, whose behavior may depend on the network's output. This dependence introduces data-dependent control flow into an otherwise fixed computation. Conventional timing analyses that track only a single interval for each program variable often cannot bound network outputs precisely enough to eliminate infeasible behaviors. As a result, they must conservatively assume the worst case for all subsequent code. Abstract interpretation can provide more precise, input-dependent bounds on network outputs and intermediate results. The technique is already widely used in the machine learning community to ensure neural network robustness. We show that the same bounds used to establish neural network robustness can be translated into flow facts to improve timing analysis. We present DeepWCET, a framework that integrates neural network abstract interpretation into the LLVM compiler toolchain and automatically generates such flow facts for WCET analysis. In a representative case study on the ACAS Xu collision avoidance neural networks, our approach reduces the analyzed WCET of the combined inference and post-inference task by up to 42 % compared to a conventional timing analysis without neural network-derived flow facts.

Keywords: Worst-case execution time (WCET) · Neural networks · Abstract interpretation · Static whole-system analysis · TriCore

1 Introduction

Neural networks are increasingly deployed in embedded safety-critical systems, particularly in the avionics and automotive domains. They enable complex physical relationships to be modeled within the limited computational resources of

a microcontroller, where conventional hand-written algorithms often fail to capture the underlying behavior. Examples range from estimating electric motor temperatures [23] to implementing policies for aircraft collision-avoidance systems [20]. In the latter case, neural networks have also been shown to provide a substantially more memory-efficient representation than traditional lookup tables, which can exceed the memory available on certifiable avionics hardware.

An essential requirement for deployment in such domains is the verification of neural network robustness. In particular, a network's decision must remain stable under small perturbations of its inputs. Today, this can be achieved with the help of abstract-interpretation-based verifiers [14]. Prominent examples include DeepZ [42], DeepPoly [43], and CROWN [54]. These methods employ relational abstract domains to model how an input region propagates through the network. The resulting bounds can be used to prove that a required property holds for all inputs within the safety-critical region. Such techniques have already found their way into recent standardization efforts. For example, ISO/PAS 8800 [19] identifies formal verification of neural network properties as one of the measures for providing evidence of robustness.

In addition to satisfying robustness requirements, safety-critical applications must also provide sound worst-case execution time (WCET) bounds, as required by standards such as ISO 26262 [18] and DO-178C [38]. In practice, however, soundness alone is not sufficient. The bounds must also be tight. Overly conservative WCET estimates waste computational resources and reduce schedulability. Tighter bounds allow more tasks, or more computation within each task, to be accommodated in the same schedule.

Obtaining a tight WCET for neural network inference itself is typically straightforward. The structure of the generated code is fixed by the network architecture and known at compile time, and all loop bounds are constant. Moreover, neural network implementations typically rely on fixed-latency operations, making their execution time largely independent of the input. The code paths taken following the inference, however, may depend on the network's output and vary in their execution time. Thus, the network introduces data-dependent control flow into an otherwise fixed-shape computation. For a given operating mode, some of these paths may be unreachable because the corresponding network outputs cannot occur. Excluding such infeasible paths from the analysis can result in a tighter mode-specific WCET and free additional execution time for other tasks. Doing so requires a sound bound on the network's possible outputs over the corresponding range of inputs.

Existing WCET tools already perform value analysis using abstract interpretation [9]. Mainstream timing analyzers [13,1] use non-relational domains such as intervals or circular linear progressions [7]. These domains track each program variable independently and therefore ignore the relationships between neurons, yielding highly pessimistic bounds for neural networks. As a result, the over-approximation accumulates from layer to layer. What is missing is a neural network-aware abstract domain that captures these inter-neuron correlations.

Our key observation is that the abstract domains developed by the machine learning community for robustness verification can also be used for WCET analysis. For each system mode, corresponding to a particular set of environmental conditions and input range, they compute tight bounds on the network's possible outputs, from which the feasible execution paths can be derived. The analysis also provides bounds on intermediate values, which can be used to bound the operands of variable-latency instructions.

In summary, this paper makes the following contributions:

1. We show how the abstract domains commonly used for neural network robustness verification can be repurposed for WCET analysis, and how the resulting bounds can be used as flow facts in a timing analysis.
2. We present DeepWCET, a whole-system framework built on the LLVM/MLIR compiler infrastructure [26,27]. It analyzes static-shape feed-forward networks in the ONNX exchange format [35] and propagates the resulting bounds to the machine-code level, where they are consumed by the timing analysis. We evaluate our framework on the TriCore [47] architecture, which is widely used in safety-critical embedded systems.
3. We provide a case study based on the widely used ACAS Xu collision-avoidance networks for unmanned aerial vehicles (UAVs) [20]. We pair the benchmark with a representative post-inference task whose WCET depends on the network's prediction, and show how tighter output bounds eliminate infeasible execution paths and reduce the resulting WCET across different operating modes.

The remainder of this paper is structured as follows. Section 2 introduces feed-forward neural networks, their abstract interpretation, and whole-system WCET analysis. Section 3 presents DeepWCET's design, and Section 4 details its implementation. Section 5 evaluates the resulting WCET improvements and analysis costs in a representative case study. Finally, Section 6 discusses related work, and Section 7 concludes with future work.

2 Background

This section introduces the neural networks we consider (Section 2.1), the abstract domains used to analyze them (Section 2.2), and the whole-system WCET analysis into which the resulting value bounds are integrated (Section 2.3).

2.1 Feed-Forward Neural Networks

The networks we consider in this work are feed-forward multi-layer perceptrons (MLPs), which are composed of a sequence of affine transformations interleaved with element-wise activation functions. During inference, an input vector is propagated through the network one layer at a time. Writing $x^{(0)} = x$ for the input, a network of L layers computes

$$x^{(k)} \;=\; \sigma\big(W^{(k)} x^{(k-1)} + b^{(k)}\big), \qquad k = 1, \ldots, L, \tag{1}$$

where $W^{(k)}$ and $b^{(k)}$ are the trained weight matrix and bias vector of layer k, and σ is the activation function, which is applied component-wise. Throughout this work, we assume $\sigma = \mathrm{ReLU}$, where $\mathrm{ReLU}(t) = \max(0, t)$ is the piecewise-linear rectified linear unit. This is the common choice for the networks in this domain, and it is the one for which all of the abstract domains of Section 2.2 are defined. The inference executes the same sequence of operations for every input. In particular, the dimensions of all intermediate vectors and matrices, and thus the resulting loop bounds, are fixed by the network architecture, leaving no input-dependent variation in the control flow.

We distinguish between regression and classification networks because the surrounding program uses their outputs differently. A *regression* network returns one or more physical quantities, that is, real-valued outputs that are consumed directly by the program. A *classification* network instead assigns one score $x_i^{(L)}$ to each class i, which reflects a cost or a probability. The predicted class is then computed as

$$c = \underset{i \in \{1,\ldots,n\}}{\arg\min}\, x_i^{(L)} \qquad (\text{or } \arg\max, \text{ depending on the convention}). \tag{2}$$

We use the $\arg\min$ convention throughout the paper because the networks in our case study select the lowest-cost class. Regression and classification networks also differ in their WCET analysis. For regression networks, the returned values must be bounded numerically because they directly influence subsequent computations. For classification networks, the prediction determines the control flow, so bounding the network's outputs allows infeasible execution paths to be excluded. The selected class is the one whose score is lower than all others (or higher, depending on the convention). Equivalently, class j is selected if and only if:

$$x_j^{(L)} - x_i^{(L)} \leq 0 \qquad \forall i \in \{1, \ldots, n\}. \tag{3}$$

2.2 Abstract Interpretation of Neural Networks

Abstract interpretation of neural networks works by propagating an abstract element that represents a whole region of inputs through the layers of the neural network, yielding a sound over-approximation of all reachable outputs. The abstract domain determines how this region is represented and manipulated during the analysis. Several such domains have been proposed in the literature [14,42,43,54]. They differ primarily in the amount of correlation they preserve between neurons. Preserving more of these relationships leads to tighter output bounds, an effect that becomes increasingly important in deeper networks because each non-linear activation introduces additional over-approximation.

Interval/Box: One Range per Neuron. The interval (or box) domain is the standard choice for general-purpose value analysis [13]. The analysis starts from an input region $[\underline{x}_1, \overline{x}_1] \times \cdots \times [\underline{x}_n, \overline{x}_n]$ derived from physical constraints, sensor ranges, or the operating mode. These intervals are propagated through the net-

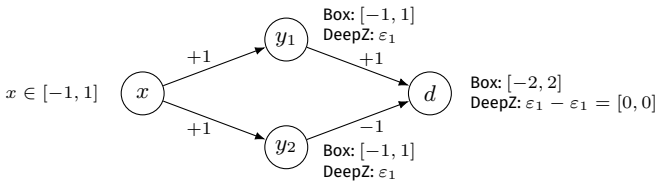


Fig. 1. Loss of correlation in the interval domain. Both hidden neurons are equal to the same input variable, so the difference $d = y_1 - y_2 = 0$. The box domain cannot capture this relation, while the noise symbols in DeepZ cancel out.

work using interval arithmetic: Affine transformations update them by multiplication and addition, while activation functions clamp the resulting bounds. The interval domain treats each neuron independently and therefore cannot capture relationships between them. As a result, the over-approximation accumulates from layer to layer and can become substantial even for moderately deep networks. Figure 1 illustrates this limitation on a simple affine computation: For $x \in [-1, 1]$ and $y_1 = y_2 = x$, the box bound on $y_1 - y_2$ is $[-2, 2]$, although the difference is always exactly 0. The domain has no way of recording that both neurons are derived from the same input, whereas the relational domains discussed next retain this information.

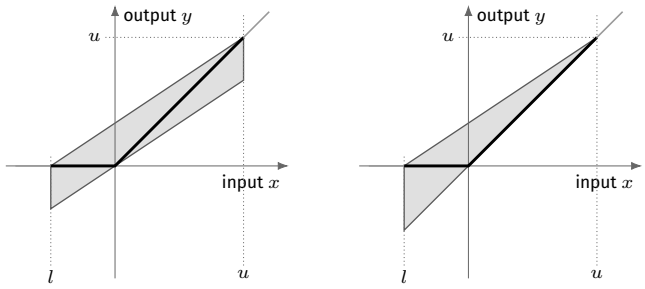
Zonotope-based Domain (DeepZ). The zonotope domain represents each neuron by an affine form

$$\hat{x} \;=\; c_0 + \sum_i c_i\, \varepsilon_i, \qquad \varepsilon_i \in [-1, 1], \tag{4}$$

over a set of shared noise symbols ε_i [42,15]. The initial noise symbols correspond to the network's input range. A concrete interval for a neuron is obtained by evaluating the affine form over all $\varepsilon_i \in [-1, 1]$, yielding $\left[c_0 - \sum_i |c_i|,\ c_0 + \sum_i |c_i|\right]$. Affine layers are handled exactly: multiplying by $W^{(k)}$ and adding $b^{(k)}$ maps affine forms to affine forms without introducing any over-approximation. Only the non-linear layers require an approximation. For a ReLU whose concrete input bounds $[l, u]$ cross zero ($l < 0 < u$), DeepZ replaces the ReLU by the minimum-area parallelogram enclosing its graph (see Figure 2a),

$$\hat{z} \;=\; \lambda \hat{x} + \mu + \mu\, \varepsilon_{\text{new}}, \qquad \lambda = \frac{u}{u - l}, \quad \mu = \frac{-l\, u}{2(u - l)}, \tag{5}$$

which introduces one fresh noise symbol ε_{new}. Stable ReLUs ($l \geq 0$ or $u \leq 0$) are represented exactly and require no additional symbol. The strength of the domain lies in the shared noise symbols. Neurons that depend on the same input retain this dependency through the common symbols, allowing correlations to be preserved across layers. For example, in Figure 1, the shared symbol ε_1 cancels in $y_1 - y_2$, yielding the exact result 0. While DeepZ retains dependencies through shared noise symbols, its symmetric relaxation of unstable ReLUs loses information that the domain discussed next can preserve more precisely.



(a) DeepZ: parallelogram (b) DeepPoly: triangle

Fig. 2. Convex relaxations of an unstable ReLU with input interval $[l, u]$. Both DeepZ and DeepPoly use the same affine upper bound. DeepZ completes the relaxation with a minimum-area parallelogram. DeepPoly instead uses a single affine lower bound, resulting in the triangular relaxation shown in (b).

Polyhedra-based Domain (DeepPoly). DeepPoly represents each neuron by a symbolic lower and upper affine bound in terms of the neurons of the preceding layer [43],

$$a_j^{\leq} \cdot x^{(k-1)} + c_j^{\leq} \;\leq\; x_j \;\leq\; a_j^{\geq} \cdot x^{(k-1)} + c_j^{\geq}, \tag{6}$$

together with concrete bounds l_j, u_j. These concrete bounds are computed by back-substitution: the symbolic bounds are repeatedly rewritten in terms of neurons from earlier layers until they are expressed solely in terms of the network inputs, at which point they are evaluated. Delaying concretization in this way preserves relationships between neurons across multiple layers. In particular, as for DeepZ, opposing symbolic terms cancel before interval bounds are introduced, avoiding much of the over-approximation incurred by the interval domain.

For an unstable ReLU with input $x \in [l, u]$ and output y, DeepPoly approximates the activation by one affine lower bound and one affine upper bound. The upper bound is the line through $(l, 0)$ and (u, u). The lower bound is chosen as either $y \geq 0$ or $y \geq x$, depending on which yields the smaller relaxation area. The resulting triangular relaxation is illustrated in Figure 2b. Unlike DeepZ, no additional symbolic variables are introduced. Instead, the computational cost is shifted to the repeated back-substitution needed to compute the concrete bounds.

Neither DeepZ nor DeepPoly dominates the other in general. Their precision depends on both the network and the verification objective [43,3]. Beyond these abstract domains, exact verification techniques based on MILP or SMT have also been proposed [46,22,21]. While complete, these approaches scale poorly even to moderately sized networks. For our purposes, relational abstract domains provide a suitable trade-off between scalability and precision. They yield sound over-approximations while typically remaining sufficiently tight for deriving useful WCET bounds. Finally, these abstract domains were originally developed for robustness verification rather than WCET analysis. In robustness verification, the computed bounds are used only to prove that a network's prediction

remains unchanged under input perturbations. Our approach instead reuses the output bounds and intermediate value bounds to constrain the program behavior considered during subsequent timing analysis.

2.3 Whole-System WCET Analysis

Implicit path enumeration (IPET) [30] formulates the control-flow graph of a program as an integer linear program (ILP), whose variables represent the execution counts of the basic blocks and whose objective is the total execution time, obtained by weighting each block with its execution cost. The constraints of this ILP are known as *flow facts*. They encode semantic information such as loop bounds, infeasible paths, or bounds on the operands of instructions. Flow facts are typically derived by a value analysis based on abstract interpretation. These constraints are crucial for obtaining tight WCET bounds. Without them, the ILP is free to choose the structurally longest path through the program, even if that path is infeasible.

Whole-system WCET analysis [12,52] extends this framework beyond individual tasks. Rather than analyzing each task in isolation, it also considers the operating system and the interactions between tasks, allowing system-wide knowledge to constrain the execution paths within each task. Our approach extends this principle to neural networks. In addition to the task set and the operating system, we analyze the neural network by abstract interpretation over the domains introduced in the previous section. The resulting bounds on neuron values and network outputs are then translated into flow facts for the same ILP formulation.

3 Design

We present DeepWCET, a framework for WCET analysis of neural network-driven applications that integrates relational abstract interpretation into the LLVM compiler infrastructure [26,27]. This section provides an overview of how DeepWCET works. It explains how value bounds are computed for neural networks using the abstract domains introduced in Section 2.2, how these bounds are propagated through the compiler, and how they are translated into flow facts to derive tight, mode-specific WCET bounds.

3.1 System Model

Our work targets safety-critical embedded systems and assumes a single-core processor without caches or timing anomalies. We further assume that timing analysis is compositional, so that a WCET bound for the entire program can be derived from the bounds of its individual basic blocks. These assumptions reflect common design choices in small, predictable safety-critical embedded systems, where hardware simplicity and timing analyzability are often prioritized over architectural complexity and performance. The programs we consider perform

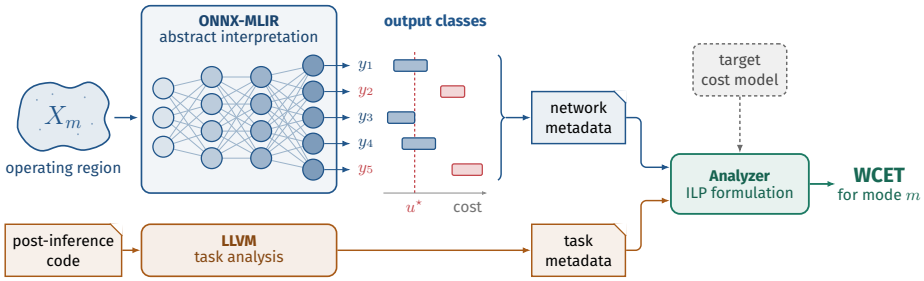


Fig. 3. Overview of DeepWCET. Given a neural network classifier and an input region X_m, DeepWCET performs abstract interpretation during compilation to compute value bounds for the network, bounding each output class y_i by a cost interval $[\ell_i, u_i]$. The classifier selects the class of lowest cost, so a class whose lower bound exceeds the smallest upper bound $u^\star = \min_i u_i$ can never be selected and is excluded (red). DeepWCET records the resulting intervals and exclusions as network metadata alongside the compiled inference code and combines them with metadata describing the post-inference task. The analyzer then translates these value bounds into flow facts and computes a mode-specific WCET.

neural network inference, whose output is subsequently consumed by one or more *post-inference* tasks. Our analysis focuses on the computations that follow the neural network inference, whose execution paths and execution times may depend on the network's output.

3.2 Overview

Figure 3 provides an overview of DeepWCET. The analysis takes as input a neural network in the ONNX exchange format [35] together with a user-defined input region. The input region is determined by the current operating mode or by assumptions on the network inputs, such as sensor ranges. Since different operating modes typically restrict the admissible inputs, each mode is analyzed separately. The ONNX model is processed by ONNX-MLIR [28], an open-source compiler framework built on LLVM Multi-Level Intermediate Representation (MLIR [27]). ONNX-MLIR lowers the network in the ONNX format through several intermediate representations before generating LLVM IR and, ultimately, executable machine code.

Our abstract interpretation is integrated into this compilation pipeline. It operates on the high-level ONNX representation, where the network structure and its operators are still explicit, and computes value bounds for the network. These bounds are attached to the corresponding compiler operations and propagated through the lowering process, establishing a mapping between the original ONNX operators and the generated LLVM IR and machine code. The resulting value bounds are emitted as metadata alongside the compiled program, following the approach of existing compiler-integrated WCET toolchains [31,17].

The post-inference code is compiled in the same way. Its program model and metadata are combined with the information obtained from the network analysis

and passed to the WCET analyzer, which derives a mode-specific timing bound for the selected input region. Annotations in the post-inference code make explicit how the network output affects the subsequent execution paths, allowing the analyzer to relate feasible network outputs to the corresponding program behavior. DeepWCET currently supports the three abstract domains introduced in Section 2.2: The interval (Box) domain, DeepZ [42], and DeepPoly [43]. In the following, we refer to the corresponding configurations of DeepWCET as $\text{DeepWCET}_{\text{Box}}$, $\text{DeepWCET}_{\text{Zono}}$, and $\text{DeepWCET}_{\text{Poly}}$, respectively. As we will show in Section 5, the choice of abstract domain has a significant impact on the precision of the inferred value bounds and, consequently, on the resulting WCET estimate.

3.3 From Network Bounds to Flow Facts

Our analysis starts from an *input region* X_m, which is determined by the current operating mode m and the corresponding assumptions on the network inputs, such as sensor value ranges. For a network with n inputs, the region is represented as one interval per input, $X_m = [\underline{x}_1, \overline{x}_1] \times \cdots \times [\underline{x}_n, \overline{x}_n]$.

DeepWCET analyzes this region using abstract interpretation and computes symbolic and concrete bounds for every intermediate neuron and network output. These bounds are then used to restrict the control flow of the post-inference task, for example by bounding the trip count of a loop that depends on the network output. This distinguishes DeepWCET from conventional robustness verifiers, which typically use the computed bounds only to establish the desired robustness property. In contrast, DeepWCET retains the intermediate and output value bounds and translates them into flow facts for the subsequent timing analysis. For regression networks, the output bounds can be used directly because the post-inference code consumes the returned values numerically.

For classification networks, the output intervals alone are not sufficient because the predicted class is determined by an arg min or arg max operation over all network outputs. Writing $f_i(x) = x_i^{(L)}$ for the score assigned to class i, the abstract interpretation computes one interval per output,

$$f_i(x) \;\in\; [\underline{x}_i, \overline{x}_i] \quad \text{for all } x \in X_m, \qquad i = 1, \ldots, n. \tag{7}$$

A class i can be excluded from the network's predictions if there exists another class j whose score is guaranteed to be lower over the entire input region, that is, if

$$\exists\, j \neq i : \qquad \max_{x \in X_m} \big(f_j(x) - f_i(x)\big) \;<\; 0. \tag{8}$$

A straightforward way to evaluate Equation (8) is to compare the concrete output intervals of Equation (7). Class i can be excluded if there exists another class j for which $\overline{x}_j < \underline{x}_i$, that is, if the entire interval of class j lies below that of class i. Figure 3 illustrates this criterion for five output classes. In practice, however, this test is often too conservative because it treats the output intervals independently and ignores the correlations between them.

Instead, for relational abstract domains, we evaluate Equation (8) symbolically by forming the difference $f_j - f_i$ from the symbolic bounds computed by the abstract interpretation. Shared symbolic terms cancel before concretization, yielding tighter bounds on the difference. The classes that satisfy Equation (8) are excluded, while the remaining classes form the set $P(X_m)$ of predictions that are feasible for the input region X_m. Because the analysis is sound, $P(X_m)$ is a sound over-approximation of the network's possible predictions. In many applications, each predicted class corresponds to a distinct execution path in the post-inference code. Excluding infeasible classes therefore allows the corresponding control-flow paths to be eliminated from the subsequent WCET analysis.

Deriving the Flow Facts. For each input region X_m, DeepWCET translates the computed network bounds into flow facts for the WCET analysis (Section 2.3). The form of these flow facts depends on how the network output is used by the program. If the prediction determines the control flow, infeasible branches are excluded from the ILP. If the output is used as a numeric value, for example as the bound of a subsequent loop, the corresponding bound is tightened instead.

More generally, let $c(i)$ denote the application-specific cost of prediction i, for example the trip count of a loop or the execution time of a particular code path. Since only the predictions in $P(X_m)$ remain feasible, the flow fact is

$$N = \max_{i \in P(X_m)} c(i). \tag{9}$$

Excluding additional predictions can only decrease this bound. If no predictions are excluded, the analysis falls back to the structural worst case.

Floating-Point Soundness. The deployed networks execute using IEEE 754 single-precision (FP32) arithmetic. Since bounds derived over real arithmetic do not necessarily hold under floating-point semantics, DeepWCET explicitly accounts for rounding errors during abstract interpretation. For each affine layer, it bounds the error of the underlying dot products using the standard floating-point error bound [16]. The resulting error is introduced as a symbolic term, allowing it to propagate relationally through subsequent layers and, where possible, cancel when forming output differences such as $f_j - f_i$ in Equation (8).

Soundness additionally requires a connection between the semantics assumed by the analysis and the machine code that is ultimately executed [45]. DeepWCET establishes this connection through its compilation pipeline. It validates the lowered module against the assumptions of the floating-point error model, including the floating-point format, operations, compiler flags, and number of roundings. If validation fails, compilation is rejected. Timing analysis without network-dependent flow facts uses the structural worst case. Together with an input region covering all admissible inputs, this ensures that $P(X_m)$ overapproximates all possible network predictions. Combined with a sound timing model, the resulting WCET bound is sound for the compiled program.

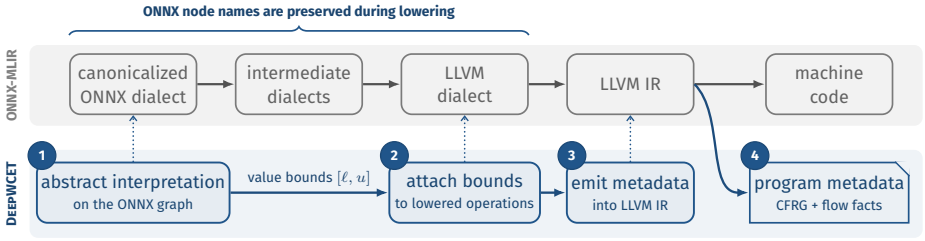


Fig. 4. Integration of DeepWCET into the ONNX-MLIR compilation pipeline. Abstract interpretation ① runs on the canonicalized ONNX dialect, where the network structure is still available. The resulting value bounds are attached to the corresponding LLVM dialect operations ②, emitted as LLVM IR metadata ③, and finally emitted as flow facts in the generated program metadata ④.

4 Implementation

DeepWCET comprises three main components: an abstract interpretation integrated into ONNX-MLIR, a metadata generation pass integrated into LLVM, and a standalone WCET analyzer. The following sections describe these three components, followed by the LLVM backend developed for our target platform.

4.1 Abstract Interpretation as a Compiler Pass

We implement the abstract interpretation as an additional analysis pass in ONNX-MLIR. The pass is enabled by a compiler flag that selects the abstract domain (`box`, `deepz`, or `deeppoly`) and takes a file describing the input region.

ONNX-MLIR lowers ONNX models to LLVM IR through a sequence of intermediate representations based on MLIR dialects, i.e., sets of operations and types tailored to a particular abstraction level (see Figure 4). Beyond the ONNX dialect, network operators are lowered to loop nests operating on memory buffers (Krnl, Affine, SCF, and MemRef dialects), at which point the network structure is no longer explicitly represented. Therefore, our pass operates at the ONNX dialect level, where both the network structure and individual operators remain explicit. It is executed after the compiler's canonicalization pass, which significantly reduces the number of abstract transformers that need to be implemented. For example, a `MatMul` operation followed by an `Add` is canonicalized into a single `Gemm` (*General matrix multiplication*) operator before our analysis is invoked.

We implemented the Box, DeepZ [42], and DeepPoly [43] domains following their original publications and reference implementations [44,53]. The implementation currently supports the basic operators required by feed-forward ReLU networks with a single, statically shaped input. If the network contains unsupported operators, no abstract interpretation is performed. The model can still be compiled, in which case the timing analysis conservatively assumes the structural worst case.

4.2 Lowering the Value Bounds to the Machine Code

The abstract interpretation operates on the ONNX dialect, whereas the generated code is produced several lowering stages later. The computed value bounds must therefore be propagated through the compilation pipeline to the LLVM backend, where the program metadata is generated.

ONNX-MLIR makes this possible through its operation naming scheme. It records the original ONNX node names in the source information of generated operations, also when several nodes are combined during compilation. These names link each ONNX node to all operations generated from it. This mapping relies on the naming convention of the ONNX-MLIR version used in this work. For each node that performs floating-point arithmetic, the validator must identify the generated operations to check their rounding errors. Compilation fails if these operations cannot be identified or if transformations, including duplication, violate the floating-point model. A reshape changes a tensor's dimensions without changing its values and may generate no additional code. The validator accepts this because reshaping introduces no rounding error. Our analysis stores the bounds as a module attribute. A second pass on the LLVM dialect uses these names to attach the previously computed bounds to the corresponding lowered operations. During translation to LLVM IR, these attributes become metadata. For classification networks, the metadata additionally stores the set of output classes that remain feasible for the selected input region.

In the LLVM backend, we reuse existing infrastructure that constructs a control-flow relation graph (CFRG) that relates LLVM IR instructions to the generated machine code [17,31]. We extended the metadata generation to emit the network-derived value bounds as flow facts alongside existing information such as constant loop bounds. The generated program metadata is consumed by the analyzer alongside the compiled binary. The post-inference code follows the same compilation pipeline, producing a second metadata file that is analyzed together with the inference metadata. Annotations make the connection between the network output and the corresponding execution paths explicit to the analyzer.

4.3 The Analyzer

The analyzer component of DeepWCET consumes the program metadata generated for the inference and post-inference code together with the corresponding binary. It is based on Platin [31], uses its metadata formats, and follows the whole-system approach of SysWCET [12]. It combines the metadata into a single ILP formulation and translates the emitted flow facts into ILP constraints. For example, if the analysis determines that certain output classes are infeasible, the analyzer excludes the corresponding control-flow paths from the ILP. Likewise, value bounds are translated into constraints on loop bounds or variable-latency instructions whenever applicable.

4.4 The TriCore Backend

The Infineon TriCore architecture [48] is widely used in safety-critical embedded systems and serves as the target platform of our evaluation. Our framework requires a native LLVM backend because the program metadata used by the timing analysis is generated during code generation. In particular, constructing the control-flow relation graph requires relating LLVM IR instructions to the emitted machine-code instructions, information that is not available when using an external compiler toolchain.

To the best of our knowledge, no actively maintained TriCore backend for a recent LLVM release is publicly available. We therefore ported an existing TriCore backend [8] to LLVM 23 and extended it to support our framework and evaluation. Existing publicly available implementations target only LLVM 2.5 or LLVM 3.7 [50]. Our implementation targets the Infineon AURIX TC3xx family, which is based on the TriCore TC1.6.2P architecture, and is intended to be compatible with the newer AURIX TC4x family. To validate the implementation, we implemented sufficient functionality to compile the Infineon Low-Level Drivers [49], the vendor-provided peripheral driver and hardware abstraction library for the AURIX platform.

The implementation follows the publicly available TriCore EABI specification [48]. Unlike most RISC architectures, TriCore distinguishes between data and address registers, and the calling convention makes use of both register classes. Implementing the backend therefore required modeling this register organization and its calling convention, including the convention that function pointers are passed in address registers. Additional work was required to support instructions operating on register pairs as well as the coexistence of 16-bit and 32-bit instruction encodings.

5 Evaluation

This section evaluates DeepWCET using the ACAS Xu collision-avoidance networks published by Julian et al. [36,20]. We first compare the supported abstract domains across the operating space and quantify the resulting WCET reductions for six representative operating modes. The analysis is then extended to all 45 ACAS Xu networks to assess the influence of different network parameters. We further compare the analyzed WCETs with measurements on the target hardware and examine the computational cost of the analysis. Finally, we study how the precision of the value bounds evolves through the network and summarize the main findings of the evaluation.

5.1 Case Study and Setup

We evaluate DeepWCET using the horizontal collision-avoidance networks of ACAS Xu for unmanned aerial systems [20]. Besides their original use in airborne collision avoidance, these networks have become one of the most widely

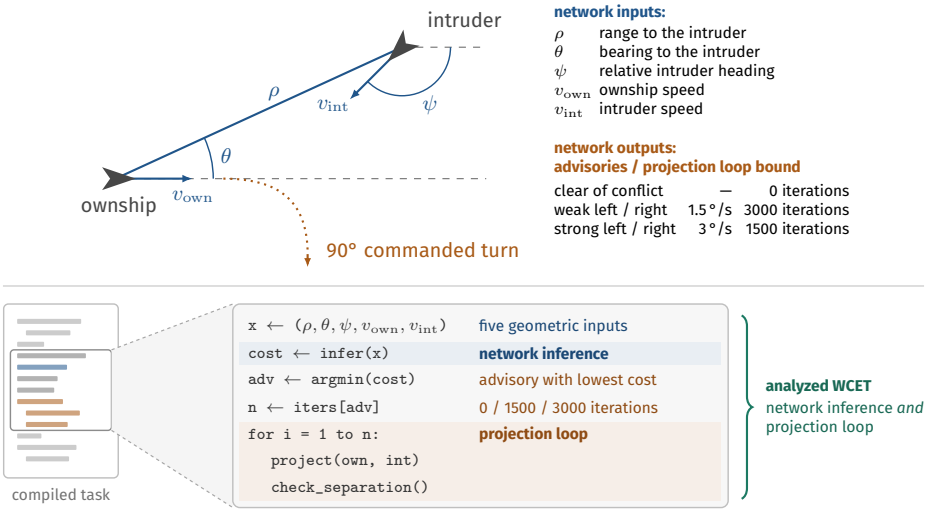


Fig. 5. Encounter geometry and analyzed task (pseudocode) of the ACAS Xu case study. The five geometric quantities define the network inputs. Depending on the selected advisory, the projection task simulates a commanded heading change of 90°, resulting in advisory-specific loop bounds.

used benchmarks for neural network robustness verification [3]. The neural networks published by Julian et al. [20] are a compressed representation of the original ACAS Xu lookup-table policy [36], reducing the required memory by approximately three orders of magnitude. For each encounter between an *ownship* and an *intruder*, the network processes five geometric features and assigns a cost to each of five advisories: weak left, strong left, clear-of-conflict, weak right, and strong right. The advisory with the lowest cost is selected. The encounter geometry and advisory set are shown in Figure 5. ACAS Xu provides 45 neural networks covering different encounter scenarios. The network is selected according to an encounter scenario and the previously issued advisory. Unless stated otherwise, we use `network_1_1`, which corresponds to the first network in this family and assumes the previous advisory to be *clear-of-conflict*.

Since ACAS Xu itself ends with the advisory selection, we integrate the network into a representative post-inference projection task, whose pseudocode is shown in the lower part of Figure 5. Given the selected advisory, the task simulates the trajectories of the ownship and intruder and repeatedly checks their predicted separation until the commanded heading change has been completed. As the pseudocode shows, the advisory is the only value passed from inference to the task and determines the trip count of the projection loop. The number of simulation steps depends on the selected advisory: strong turns require 1500 iterations, weak turns 3000 iterations, and no iterations are performed for the clear-of-conflict advisory. These iteration counts are derived from the turn rates defined by the ACAS Xu action set (3 °/s for strong turns and 1.5 °/s for weak turns), assuming a total heading change of 90° and a simulation frequency of

50 Hz. Similar projection tasks are employed by the DAIDALUS reference implementation to verify that an advisory maintains a safe separation from the intruder aircraft, as required for operating unmanned aircraft in civil airspace [34].

The target platform for all evaluations is an Infineon AURIX TC397XE running at 100 MHz. The network code and data reside in the core-local scratchpad memory, and both the inference and post-inference tasks execute on the same core, eliminating all timing anomalies except pipelining effects. Our timing model conservatively sums the worst-case instruction latencies documented in the TriCore architecture manual [47] and does not model the overlap between the core's parallel execution pipelines. The analyzed task comprises inference, advisory selection, and projection, and executes without preemption.

The ACAS Xu networks are provided in the ONNX format and can therefore be compiled and analyzed by DeepWCET without modification. We compile them with ONNX-MLIR based on MLIR 20 and our TriCore backend based on LLVM 23, using optimization level `-O2` for the network and `-O1` for the post-inference task. We measure execution times non-intrusively using a cycle-accurate on-board instruction tracer. The analyzed object code is linked with a small, statically configured OSEK-style operating system kernel, which executes the task as its only task without interrupts.

5.2 Advisory Exclusion by Abstract Domain

In this evaluation, we compare the three configurations of DeepWCET with respect to the advisories they can exclude over the operating space. We center the input regions on encounters in which both aircraft travel at 152 m/s, with a relative intruder heading of 180°. The two inputs varied across the grid are the horizontal range (150 m to 6000 m) and the relative bearing (−180° to 180°), that is, the angle between the ownship's heading and the line of sight to the intruder. The selected advisory directly determines the loop bound of the post-inference projection task: weak turns require 3000 iterations, strong turns 1500, and the clear-of-conflict advisory skips the loop entirely.

Figure 6 compares the loop bounds achievable with the three configurations with the loop bound found by sampling. Each cell represents an input region spanning 100 m in range and 6.1° in bearing. For comparison, we obtain an observed lower bound by evaluating the network at all corners and at 600 uniformly distributed interior points of each region.

The sampling results show that strong-turn advisories occur primarily for head-on encounters (0°) at short range, yielding a loop bound of 1500 iterations. At bearing angles beyond approximately 27° and distances greater than 1.1 km, the network predominantly issues the clear-of-conflict advisory. DeepWCET$_{\text{Box}}$ always produces the structural worst-case loop bound of 3000 iterations. Since the interval domain cannot exclude any advisory, it must always assume the most expensive weak-turn case. This bound nevertheless matches the sampling result in 18.2 % of all cells, namely where a weak turn is indeed required.

Both DeepWCET$_{\text{Zono}}$ and DeepWCET$_{\text{Poly}}$ are considerably more precise, as they can exclude advisories that are impossible within an input region. Com-

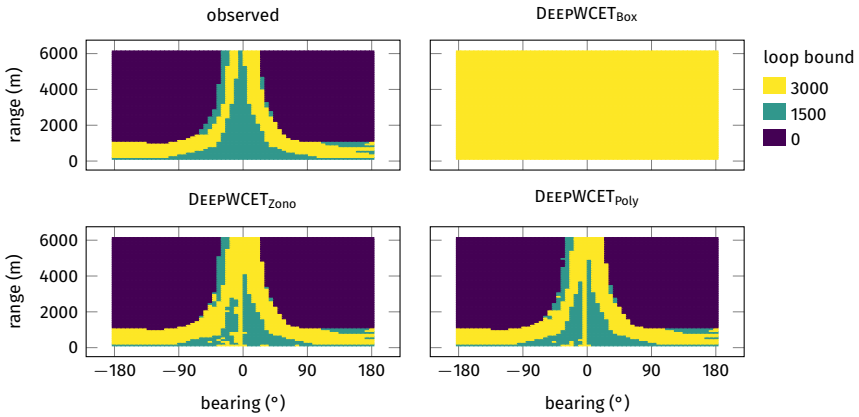


Fig. 6. Loop bounds of the post-inference projection task over the operating space. Each of the 60×60 cells represents an input region spanning 100 m in range and 6.1° in bearing. The observed result is obtained by sampling, whereas DeepWCET$_{\text{Box}}$, DeepWCET$_{\text{Zono}}$, and DeepWCET$_{\text{Poly}}$ derive sound bounds by abstract interpretation.

pared to the network-agnostic analysis, which always yields a loop bound of 3000 iterations, DeepWCET$_{\text{Zono}}$ derives a smaller sound bound in 76 % of all cells. This bound matches the sampling result in 94 % of the cells and is otherwise more conservative. DeepWCET$_{\text{Poly}}$ further improves these numbers to 78 % and 96 %, respectively. Most of this additional improvement comes from regions in which it can rule out weak-turn advisories, particularly within approximately 45° of the ownship's heading.

The precision of the computed bounds also depends on the size of the analyzed input regions. Larger regions require fewer analyses to cover the operating space, but may admit additional advisories and consequently yield looser abstract bounds. To evaluate this trade-off, we increased all five input interval widths by a factor of 4. Under these conditions, DeepWCET$_{\text{Poly}}$ still reduces the loop bound in 56 % of the regions. Further enlargement, however, substantially reduces the fraction of regions for which the analysis yields an improvement. Backward abstract interpretation methods [25,55] could help identify larger input regions that preserve the flow facts needed for tighter WCET bounds.

The results in Figure 6 may depend on the parameters of the particular network considered. We therefore repeated the analysis for all 3600 cells across each of the 45 ACAS Xu networks, which share the same architecture but differ in their weights. Figure 7 shows, per network and configuration, the fraction of cells whose bound is improved, split by whether the projection is skipped entirely (0 iterations) or cut to 1500 iterations. DeepWCET$_{\text{Box}}$ improves no cell of any network. On average, DeepWCET$_{\text{Zono}}$ and DeepWCET$_{\text{Poly}}$ reduce the loop bound in 76.6 % and 78.2 % of cells, respectively. Across the network family, DeepWCET$_{\text{Poly}}$ reduces the bound in between 60.9 % and 100 % of cells.

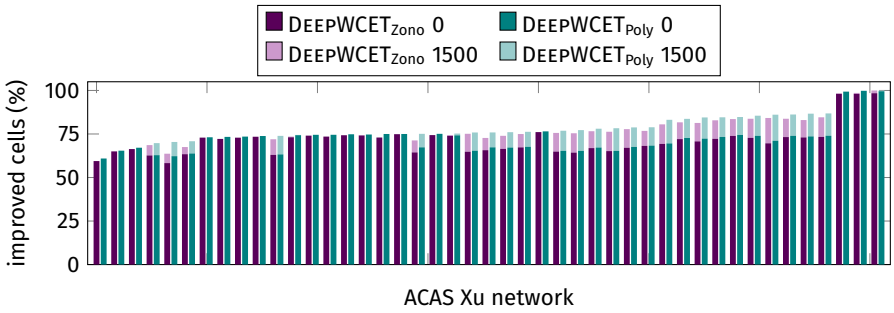


Fig. 7. Fraction of cells for all 45 ACAS Xu networks, where DeepWCET$_{\text{Zono}}$ and DeepWCET$_{\text{Poly}}$ could lower the projection loop bound to 1500 or 0. Each network is analyzed over the 60×60 grid of Figure 6. Each pair represents one network, ordered by increasing fraction of cells improved by DeepWCET$_{\text{Poly}}$.

Table 1. Advisory exclusion and resulting WCET reductions for six representative operating modes. The speed of both aircraft varies between 130 m/s to 190 m/s. The relative WCETs are normalized to the network-agnostic bound of 4597 µs.

Mode and input region	Configuration	Excluded	Loop bound	Rel. WCET
behind, opposite heading	DeepWCET$_{\text{Zono}}$	4	0	0.58×
$\rho \in [6218, 12070]$ m, $\theta \in [149, 180]°$, $\psi \in [164, 180]°$	DeepWCET$_{\text{Poly}}$	4	0	0.58×
right side, passing	DeepWCET$_{\text{Zono}}$	4	0	0.58×
$\rho \in [7681, 10607]$ m, $\theta \in [-106, -74]°$, $\psi \in [176, 180]°$	DeepWCET$_{\text{Poly}}$	4	0	0.58×
behind right, tight sector	DeepWCET$_{\text{Zono}}$	4	0	0.58×
$\rho \in [5462, 6730]$ m, $\theta \in [-180, -167]°$, $\psi \in [176, 180]°$	DeepWCET$_{\text{Poly}}$	4	0	0.58×
behind right, wide sector	DeepWCET$_{\text{Zono}}$	0	3000	1.00×
$\rho \in [2292, 9900]$ m, $\theta \in [-180, -134]°$, $\psi \in [166, 180]°$	DeepWCET$_{\text{Poly}}$	4	0	0.58×
behind, close	DeepWCET$_{\text{Zono}}$	3	1500	0.79×
$\rho \in [1173, 1265]$ m, $\theta \in [144, 156]°$, $\psi \in [177, 180]°$	DeepWCET$_{\text{Poly}}$	3	1500	0.79×
approaching head-on	DeepWCET$_{\text{Zono}}$	0	3000	1.00×
$\rho \in [3962, 5791]$ m, $\theta \in [-10, 10]°$, $\psi \in [165, 180]°$	DeepWCET$_{\text{Poly}}$	0	3000	1.00×

5.3 Mode-Specific WCET Analysis

To illustrate the WCET reductions achievable for this case study, we select six representative operating modes and compute their WCET. The corresponding input regions and analysis results are summarized in Table 1. The reported input regions were obtained by iteratively expanding representative operating points, one input dimension at a time, until the abstract domain lost the ability to rule out additional advisories. The network-agnostic analysis provides the baseline and always assumes the maximum loop bound of 3000 iterations, resulting in a WCET of 4597 µs. Since DeepWCET$_{\text{Box}}$ cannot exclude any advisory in these regions, it produces the same result.

Consistent with the previous evaluation, the relational configurations achieve the largest WCET reductions. For three of the six operating modes, both configurations reduce the WCET of the network inference and post-inference task together to 0.58× compared to the network-agnostic baseline, which always as-

Table 2. Analyzed WCET and worst-observed execution time of the combined inference, advisory selection, and projection task for each loop bound. A bound of 0 skips the projection loop. Each observed time is the maximum ± standard deviation of 30 measurements obtained from cycle-accurate instruction traces.

Loop bound	Worst-observed (µs)	WCET (µs)	Overapprox.
0	1258.98 ± 0.000	2676.83	2.13×
1500	1949.04 ± 0.013	3636.83	1.87×
3000	2638.99 ± 0.005	4596.83	1.74×

sumes the maximum loop bound. In the (*behind, close*) mode, both weak-turn advisories but only one strong-turn advisory (in one direction) can be excluded, reducing the loop bound to 1500 iterations and the WCET to 0.79× of the baseline. The (*behind right, wide sector*) mode is obtained by expanding the (*behind right, tight sector*) input region by a factor of six in range and 3.5 in each of the two angles. It is the only mode where the relational configurations differ: DeepWCET$_{\text{Poly}}$ still proves that only the clear-of-conflict advisory is feasible, whereas DeepWCET$_{\text{Zono}}$ falls back to the interval result. We confirmed DeepWCET$_{\text{Poly}}$'s result by sampling the enlarged input region, which produced only the clear-of-conflict advisory. The remaining operating mode (*approaching, head-on*) shows no achievable WCET reductions, which is expected as the intruder is heading towards the ownship. In contrast, DeepWCET$_{\text{Box}}$ never excludes any advisories for these operating modes and therefore always produces the same WCET as the network-agnostic baseline.

Table 2 compares the analyzed WCETs with measurements on the target hardware, using six different network inputs from the *approaching, head-on* mode that together cover all advisories (one for loop bound 0, two for 1500, and three for 3000). In all cases, the measured execution times remain below the respective bounds, with overestimation factors between 1.74× and 2.13×. Since code and data reside in scratchpad memory and the task runs without preemption, executions with the same loop bound differ by at most 7 cycles. Comparing the analyzed WCETs with the traces shows that the overestimation mainly stems from overlapping instructions. Between 44 and 60 % of instructions complete in the same cycle as at least one other instruction, such as a load and a floating-point multiplication. However, our cost model accounts for these instructions sequentially, thereby overestimating their combined execution time. A more detailed model of the TC1.6.2P core's integer, load/store, and loop pipelines could therefore substantially tighten the WCET bounds.

5.4 Analysis Precision and Cost

We validate the sampling results on 400 randomly selected cells from Figure 6 using branch and bound [4], recursively splitting each region until the sampled and verified advisory sets coincide. This establishes the exact advisory set for 96.5 % of the cells, in all cases confirming the original sampling result.

We additionally evaluate the computational cost of the abstract interpretation. All analysis times were measured on a laptop with an AMD Ryzen 7 PRO

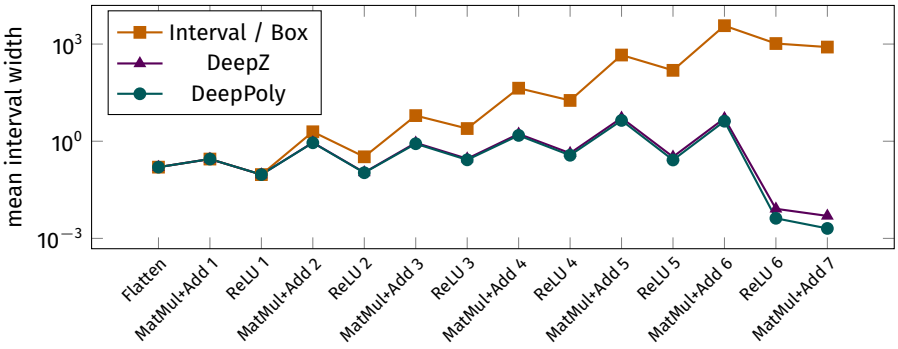


Fig. 8. Mean interval width after each network operation for the *behind, opposite heading* operating mode of Table 1. The operations are shown in graph order. Note the logarithmic y-axis scaling.

8840HS processor. For ACAS Xu, analyzing one input region takes approximately 0.19 s per domain, compared with 1.05 s for compilation and 0.90 s for solving the ILP formulation. To assess scaling beyond ACAS Xu, we also analyze fully connected MNIST networks from the ERAN benchmark suite [44], with 3 to 9 affine layers and widths from 50 to 200. On these networks, DeepWCET$_{\text{Poly}}$ takes 3.1× as long as DeepWCET$_{\text{Box}}$ and 1.3× as long as DeepWCET$_{\text{Zono}}$ on average. Its analysis of the largest network, with 440 410 parameters, takes 3.16 s. Analysis times increase with network depth and width, while networks with the same architecture but different weights show similar runtimes. Overall, relational analysis remains practical for networks substantially larger than those considered in our case study.

5.5 Propagation of Value Bounds

To examine how the precision of the computed value bounds changes throughout the network, we measure the mean interval width after every network operation for the *behind, opposite heading* operating mode. Figure 8 shows these widths in graph order on a logarithmic scale.

The figure explains why the interval domain could not exclude any advisories in the previous evaluations. Its value bounds grow rapidly throughout the network, exceeding those of DeepPoly by more than five orders of magnitude at the output. This loss of precision results from treating neurons independently, causing dependencies introduced by affine layers to be discarded after every operation. The interval widths of DeepZ and DeepPoly remain close throughout the network, indicating that both relational domains retain considerably more precision than the interval domain. DeepPoly nevertheless produces slightly tighter bounds. This is consistent with the different ReLU relaxations used by the two domains. Whereas DeepZ introduces a symmetric error term for uncertain ReLUs, DeepPoly maintains separate lower and upper affine bounds and exploits them during back-substitution.

5.6 Applicability and Limitations

DeepWCET applies whenever the network's output influences the execution time of the subsequent code. It only requires a mapping from each feasible output to a cost $c(i)$ in Equation (9), such as a loop bound, the cost of a selected branch or function, or an entry of a parameter table. In model predictive control [24,40], for example, a neural network often provides the warm start of an iterative solver, whose iteration count depends on how far this warm start is from the optimal solution. Bounding the network output over an input region bounds this distance, from which a maximum iteration count for the solver loop can be derived.

Our implementation currently supports feed-forward ReLU networks, but DeepZ and DeepPoly also support convolutions, sigmoid, and tanh, and DeepPoly additionally supports max-pooling [42,43]. Extending our implementation to these operators would mainly involve integrating the existing abstract transformers into ONNX-MLIR. Beyond extending operator support, the approach could also be applied to inference on GPUs and AI accelerators. This would require a sound hardware timing model and a floating-point error model that accounts for the target hardware's precision and accumulation order.

5.7 Summary

Our case study demonstrates that relational abstract domains can significantly reduce the WCET of post-inference tasks by excluding advisories that are infeasible within a given input region. In our evaluation, this either eliminates the projection loop entirely or reduces its iteration count from 3000 to 1500, yielding a WCET of up to 0.58× that of the network-agnostic baseline.

Two factors determine the effectiveness of this approach. First, the choice of abstract domain is crucial: DeepWCET$_{\text{Box}}$ does not improve the WCET, whereas the relational configurations exclude infeasible advisories over large parts of the operating space. In our evaluation, DeepWCET$_{\text{Poly}}$ yields bounds at least as tight as DeepWCET$_{\text{Zono}}$ in almost all cases and strictly tighter in several, with moderate additional analysis cost. We therefore recommend DeepWCET$_{\text{Poly}}$ as the default configuration. Second, the achievable reduction depends strongly on the operating mode. The choice of input regions is therefore an important part of applying DeepWCET, as separate flow facts and WCET bounds are derived for each region.

6 Related Work

Abstract domains for neural networks. Certifying neural networks by abstract interpretation was introduced by AI^2 [14]. Among the many abstract domains proposed since then, DeepZ [42] and DeepPoly [43] remain the standard single-neuron relaxations and are the domains used in DeepWCET. CROWN [54] derives equivalent bounds for ReLU networks using a different formulation [39]. Since CROWN and DeepPoly coincide for the networks considered in this work when using the same lower relaxation, we only implemented DeepPoly.

Beyond the single-neuron relaxations, more precise output bounds can be achieved. Multi-neuron relaxations bound groups of activations jointly [41,33]. By capturing dependencies between several neurons, these methods improve precision at the cost of substantially higher analysis times, typically requiring linear programming. Exact verification methods formulate the network as a mixed-integer program [46] or SMT problem [22], making them even more expensive. The annual VNN-COMP tracks where the resulting trade-offs currently stand [3]. For WCET analysis, however, the tightest possible output bounds are not required. It is sufficient to exclude infeasible execution paths, which our evaluation shows can already be achieved using DeepPoly.

Mode-specific and parametric WCET. Prior work has explored deriving WCET bounds for different operating conditions rather than a single bound for the entire program. Parametric WCET analysis expresses the WCET as a function of program parameters, using approaches based on abstract interpretation and parametric integer programming [6] or symbolic computation of WCET formulas [2]. Mixed-criticality systems also associate multiple WCET estimates with a task, but these correspond to different levels of assurance rather than different operating conditions [51,5]. Our approach is more closely related to parametric WCET, as DeepWCET likewise derives WCET bounds that depend on program inputs. Rather than expressing the WCET symbolically as a function of program parameters, however, DeepWCET uses abstract interpretation to propagate a user-defined input region through the neural network and derive output bounds that restrict feasible execution paths after inference.

Timing analysis of neural network workloads. Prior work on neural network timing analysis has primarily focused on bounding the execution time of inference. ACETONE generates semantics-preserving C code from trained feed-forward networks whose WCET can be analyzed using the OTAWA WCET analysis framework [10], with later work improving inference performance through predictable GEMM-based convolution routines [11]. Like DeepWCET, ACETONE exploits the structure of the network for timing analysis. However, ACETONE bounds the execution time of the inference itself, whereas DeepWCET derives bounds on network outputs and propagates them as flow facts to improve the WCET analysis of the surrounding code. Other works have investigated predictability at the hardware and system levels. Restuccia and Biondi analyze worst-case response times for DNN inference on FPGA SoCs and propose hardware support for bounding accelerator interference [37]. A recent survey likewise identifies memory hierarchies, accelerator arbitration, and scheduling as important challenges for predictable neural network inference [29].

7 Conclusion and Future Work

Feed-forward networks turn a computation of fixed shape into data-dependent control flow: the inference itself costs the same on every input, while the post-inference code that consumes its results does not. This paper has shown that the abstract domains built by the machine-learning community to certify robustness can be reused by timing analyses to bound the WCET of post-inference tasks, no new domains are needed. By bounding the network's output over a pre-defined input region in an operating mode, certain output classes can be excluded, which translates to flow facts in the resulting ILP and finally to a reduced WCET estimate. We implemented this approach end to end in DeepWCET, integrating neural network value analysis into the ONNX-MLIR compilation pipeline and lowering the resulting bounds to the machine code consumed by the WCET analyzer.

Our case study on the ACAS Xu networks showed that relational abstract domains can significantly reduce per-mode WCETs. Depending on the operating mode, the WCET of the combined inference and projection task was reduced to 0.79× and 0.58× of the network-agnostic baseline by excluding advisories that cannot occur within the analyzed input region. Although the achievable reduction depends on the application and the chosen operating regions, DeepPoly and DeepZ tighten the network output bounds by orders of magnitude compared to the interval domain traditionally used in WCET analysis.

The presented approach is not limited to ACAS Xu and is applicable wherever neural network outputs influence the execution time of subsequent computations. One promising application is model predictive control, where neural networks are used to warm-start iterative optimization algorithms [24,40]. Since the network prediction affects the number of solver iterations, the same approach could be used to derive tighter WCET bounds for the optimization stage.

DeepWCET benefits from tight regions per operating mode. In the current implementation, operating regions are provided by the developer based on the application's operating modes and relevant sensor ranges. An interesting direction is to infer these regions automatically using backward abstract interpretation methods [25,55], allowing the largest regions that preserve the required flow facts to be determined automatically.

Finally, this work considered feed-forward networks only. Implicit neural networks, such as deep-equilibrium models, define their outputs as fixed points of iterative computations, thereby introducing loops into the inference process itself. Existing abstract domains for these models [32] provide a natural starting point for extending DeepWCET to derive WCET bounds for such architectures.

Acknowledgments. This work has been supported and funded by the Schaeffler Hub for Advanced Research at the Friedrich-Alexander-Universität Erlangen-Nürnberg (SHARE at FAU).

Disclosure of Interests. Apart from the support acknowledged above, the authors have no further competing interests to declare that are relevant to the content of this article.

References

1. Ballabriga, C., Cassé, H., Rochange, C., Pascal, S.: OTAWA: An open toolbox for adaptive WCET analysis. In: Software Technologies for Embedded and Ubiquitous Systems (2010). https://doi.org/10.1007/978-3-642-16256-5_6
2. Ballabriga, C., Forget, J., Lipari, G.: Symbolic WCET computation. ACM Transactions on Embedded Computing Systems (TECS), Volume 17, Issue 2 (2017). https://doi.org/10.1145/3147413
3. Brix, C., Müller, M.N., Bak, S., Johnson, T.T., Liu, C.: First three years of the international verification of neural networks competition (VNN-COMP). International Journal on Software Tools for Technology Transfer (2023). https://doi.org/10.1007/s10009-023-00703-4
4. Bunel, R., Turkaslan, I., Torr, P.H.S., Kumar, M.P., Lu, J., Kohli, P.: Branch and bound for piecewise linear neural network verification. The Journal of Machine Learning Research, Volume 21, Issue 1 (2020), https://dl.acm.org/doi/10.5555/3455716.3455758
5. Burns, A., Davis, R.I.: A survey of research into mixed criticality systems. ACM Computing Surveys (CSUR), Volume 50, Issue 6 (2017). https://doi.org/10.1145/3131347
6. Bygde, S., Lisper, B.: Towards an automatic parametric WCET analysis. In: 8th International Workshop on Worst-Case Execution Time Analysis (WCET'08) (2008). https://doi.org/10.4230/OASIcs.WCET.2008.1659
7. Cassé, H., Birée, F., Sainrat, P.: Multi-architecture Value Analysis for Machine Code. In: 13th International Workshop on Worst-Case Execution Time Analysis (2013). https://doi.org/10.4230/OASIcs.WCET.2013.42
8. Christoph Erhardt: LLVM with TriCore support, https://gitlab.cs.fau.de/sicherha/llvm, Accessed 2026-07-31
9. Cousot, P., Cousot, R.: Abstract interpretation: a unified lattice model for static analysis of programs by construction or approximation of fixpoints. In: POPL '77: Proceedings of the 4th ACM SIGACT-SIGPLAN symposium on Principles of programming languages (1977). https://doi.org/10.1145/512950.512973
10. De Albuquerque Silva, I., Carle, T., Gauffriau, A., Pagetti, C.: ACETONE: Predictable Programming Framework for ML Applications in Safety-Critical Systems. In: 34th Euromicro Conference on Real-Time Systems (ECRTS 2022) (2022). https://doi.org/10.4230/LIPIcs.ECRTS.2022.3
11. De Albuquerque Silva, I., Carle, T., Gauffriau, A., Pagetti, C.: Extending a predictable machine learning framework with efficient gemm-based convolution routines. Real-Time Systems (2023). https://doi.org/10.1007/s11241-023-09407-z
12. Dietrich, C., Wägemann, P., Ulbrich, P., Lohmann, D.: SysWCET: Whole-system response-time analysis for fixed-priority real-time systems (outstanding paper). In: 2017 IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS) (2017). https://doi.org/10.1109/RTAS.2017.37
13. Ferdinand, C., Heckmann, R.: aiT: Worst-case execution time prediction by static program analysis. In: Building the Information Society (2004). https://doi.org/10.1007/978-1-4020-8157-6_29
14. Gehr, T., Mirman, M., Drachsler-Cohen, D., Tsankov, P., Chaudhuri, S., Vechev, M.: AI2: Safety and robustness certification of neural networks with abstract interpretation. In: 2018 IEEE Symposium on Security and Privacy (SP) (2018). https://doi.org/10.1109/SP.2018.00058

15. Ghorbal, K., Goubault, E., Putot, S.: The zonotope abstract domain taylor1+. In: Computer Aided Verification (2009). https://doi.org/10.1007/978-3-642-02658-4_47
16. Higham, N.J.: Accuracy and Stability of Numerical Algorithms. Society for Industrial and Applied Mathematics, 2 edn. (2002). https://doi.org/10.1137/1.9780898718027, Sec. 3.1, p. 63
17. Huber, B., Prokesch, D., Puschner, P.: Combined WCET analysis of bitcode and machine code using control-flow relation graphs. In: LCTES '13: Proceedings of the 14th ACM SIGPLAN/SIGBED conference on Languages, compilers and tools for embedded systems (2013). https://doi.org/10.1145/2491899.2465567
18. International Organization for Standardization: ISO 26262: Road vehicles — Functional safety (2018), https://www.iso.org/standard/68383.html, Accessed 2026-07-31
19. International Organization for Standardization: ISO/PAS 8800: Road vehicles — Safety and artificial intelligence (2024), https://www.iso.org/standard/83303.html
20. Julian, K.D., Kochenderfer, M.J., Owen, M.P.: Deep neural network compression for aircraft collision avoidance systems. Journal of Guidance, Control, and Dynamics (2019). https://doi.org/10.2514/1.G003724
21. Katz, G., Barrett, C., Dill, D.L., Julian, K., Kochenderfer, M.J.: Reluplex: An efficient SMT solver for verifying deep neural networks. In: Computer Aided Verification (2017). https://doi.org/10.1007/978-3-319-63387-9_5
22. Katz, G., Huang, D.A., Ibeling, D., Julian, K., Lazarus, C., Lim, R., Shah, P., Thakoor, S., Wu, H., Zeljić, A., Dill, D.L., Kochenderfer, M.J., Barrett, C.: The marabou framework for verification and analysis of deep neural networks. In: Computer Aided Verification (2019). https://doi.org/10.1007/978-3-030-25540-4_26
23. Kirchgässner, W., Wallscheid, O., Böcker, J.: Estimating electric motor temperatures with deep residual machine learning. IEEE Transactions on Power Electronics (TPEL) (2021). https://doi.org/10.1109/TPEL.2020.3045596
24. Klaučo, M., Kalúz, M., Kvasnica, M.: Machine learning-based warm starting of active set methods in embedded model predictive control. Engineering Applications of Artificial Intelligence (2019). https://doi.org/10.1016/j.engappai.2018.09.014
25. Kotha, S., Brix, C., Kolter, Z., Dvijotham, K.D., Zhang, H.: Provably bounding neural network preimages. In: NIPS '23: Proceedings of the 37th International Conference on Neural Information Processing System (2023), https://dl.acm.org/doi/10.5555/3666122.3669640
26. Lattner, C., Adve, V.: LLVM: A compilation framework for lifelong program analysis & transformation. In: CGO '04: Proceedings of the international symposium on Code generation and optimization: feedback-directed and runtime optimization (2004), https://dl.acm.org/doi/10.5555/977395.977673
27. Lattner, C., Amini, M., Bondhugula, U., Cohen, A., Davis, A., Pienaar, J., Riddle, R., Shpeisman, T., Vasilache, N., Zinenko, O.: MLIR: Scaling compiler infrastructure for domain specific computation. In: 2021 IEEE/ACM International Symposium on Code Generation and Optimization (CGO) (2021). https://doi.org/10.1109/CGO51591.2021.9370308
28. Le, T.D., Bercea, G., Chen, T., Eichenberger, A.E., Imai, H., Jin, T., Kawachiya, K., Negishi, Y., O'Brien, K.: Compiling ONNX neural network models using MLIR. Computing Research Repository (2020), https://arxiv.org/abs/2008.08272

29. Lee, S., Kang, W., Bertogna, M., Chwa, H.S., Lee, J.: Timing guarantees for inference of AI models in embedded systems. Real-Time Systems (2025). https://doi.org/10.1007/s11241-025-09445-9
30. Li, Y.T.S., Malik, S.: Performance analysis of embedded software using implicit path enumeration. In: Proceedings of the 32nd Annual ACM/IEEE Design Automation Conference (1995). https://doi.org/10.1145/217474.217570
31. Maroun, E.J., Dengler, E., Dietrich, C., Hepp, S., Herzog, H., Huber, B., Knoop, J., Wiltsche-Prokesch, D., Puschner, P., Raffeck, P., Schoeberl, M., Schuster, S., Wägemann, P.: The platin multi-target worst-case analysis tool. In: 22nd International Workshop on Worst-Case Execution Time Analysis (WCET 2024) (2024). https://doi.org/10.4230/OASIcs.WCET.2024.2
32. Müller, M.N., Fischer, M., Staab, R., Vechev, M.: Abstract interpretation of fixpoint iterators with applications to neural networks. Proceedings of the ACM on Programming Languages, Volume 7, Issue PLDI (2023). https://doi.org/10.1145/3591252
33. Müller, M.N., Makarchuk, G., Singh, G., Püschel, M., Vechev, M.: PRIMA: general and precise neural network certification via scalable convex hull approximations. In: Proceedings of the ACM on Programming Languages, Volume 6, Issue POPL (2022). https://doi.org/10.1145/3498704
34. Muñoz, C., Narkawicz, A., Hagen, G., Upchurch, J., Dutle, A., Consiglio, M., Chamberlain, J.: DAIDALUS: Detect and avoid alerting logic for unmanned systems. In: 2015 IEEE/AIAA 34th Digital Avionics Systems Conference (DASC) (2015). https://doi.org/10.1109/DASC.2015.7311421
35. ONNX: Open Neural Network Exchange, https://onnx.ai, Accessed 2026-07-31
36. Owen, M.P., Panken, A., Moss, R., Alvarez, L., Leeper, C.: ACAS Xu: Integrated collision avoidance and detect and avoid capability for UAS. In: 2019 IEEE/AIAA 38th Digital Avionics Systems Conference (DASC) (2019). https://doi.org/10.1109/DASC43569.2019.9081758
37. Restuccia, F., Biondi, A.: Time-predictable acceleration of deep neural networks on FPGA SoC platforms. In: 2021 IEEE Real-Time Systems Symposium (RTSS) (2021). https://doi.org/10.1109/RTSS52674.2021.00047
38. RTCA, Inc.: DO-178C, Software Considerations in Airborne Systems and Equipment Certification (2012)
39. Salman, H., Yang, G., Zhang, H., Hsieh, C.J., Zhang, P.: A convex relaxation barrier to tight robustness verification of neural networks. In: Proceedings of the 33rd International Conference on Neural Information Processing Systems (2019), https://dl.acm.org/doi/10.5555/3454287.3455169
40. Sambharya, R., Hall, G., Amos, B., Stellato, B.: End-to-end learning to warm-start for real-time quadratic optimization. In: Proceedings of the 5th Annual Learning for Dynamics and Control Conference (2023), https://proceedings.mlr.press/v211/sambharya23a.html
41. Singh, G., Ganvir, R., Püschel, M., Vechev, M.: Beyond the single neuron convex barrier for neural network certification. In: Proceedings of the 33rd International Conference on Neural Information Processing Systems (2019), https://dl.acm.org/doi/10.5555/3454287.3455639
42. Singh, G., Gehr, T., Mirman, M., Püschel, M., Vechev, M.: Fast and effective robustness certification. In: NIPS'18: Proceedings of the 32nd International Conference on Neural Information Processing Systems (2018), https://dl.acm.org/doi/10.5555/3327546.3327739

43. Singh, G., Gehr, T., Püschel, M., Vechev, M.: An abstract domain for certifying neural networks. In: Proceedings of the ACM on Programming Languages, Volume 3, Issue POPL (2019). https://doi.org/10.1145/3290354
44. SRI Lab: ERAN: ETH Robustness Analyzer for Neural Networks, https://github.com/eth-sri/eran, Accessed 2026-07-31
45. Szász, A., Bánhelyi, B., Jelasity, M.: No soundness in the real world: On the challenges of the verification of deployed neural networks. In: Proceedings of the 42nd International Conference on Machine Learning (2025), https://proceedings.mlr.press/v267/szasz25a.html
46. Tjeng, V., Xiao, K.Y., Tedrake, R.: Evaluating robustness of neural networks with mixed integer programming. In: 7th International Conference on Learning Representations, ICLR 2019 (2019), https://openreview.net/forum?id=HyGIdiRqtm
47. AURIX TC3xx user manual, https://web.archive.org/web/20260804123350/https://documentation.infineon.com/aurixtc3xx/docs/qmd1702366622648
48. Embedded Applications Binary Interface (EABI): V1.3x and V1.6x TriCore Architectures, https://web.archive.org/web/20260806060433/https://ba.ic-components.com/files/f3/SAK-TC264DE-40F200N%20BC.pdf
49. iLLD Release TC3x, https://github.com/Infineon/illd_release_tc3x, accessed 2026-07-31
50. TriDis: llvm-tricore: LLVM Compiler Infrastructure with TriCore backend, https://github.com/TriDis/llvm-tricore, Accessed 2026-07-31
51. Vestal, S.: Preemptive scheduling of multi-criticality systems with varying degrees of execution time assurance. In: 28th IEEE International Real-Time Systems Symposium (RTSS 2007) (2007). https://doi.org/10.1109/RTSS.2007.47
52. Wägemann, P., Dietrich, C., Distler, T., Ulbrich, P., Schröder-Preikschat, W.: Whole-system worst-case energy-consumption analysis for energy-constrained real-time systems. In: 30th Euromicro Conference on Real-Time Systems (ECRTS 2018) (2018). https://doi.org/10.4230/LIPIcs.ECRTS.2018.24
53. Xu, K., Shi, Z., Zhang, H., Wang, Y., Chang, K.W., Huang, M., Kailkhura, B., Lin, X., Hsieh, C.J.: Automatic perturbation analysis for scalable certified robustness and beyond. In: NIPS'20: Proceedings of the 34th International Conference on Neural Information Processing Systems (2020), https://dl.acm.org/doi/10.5555/3495724.3495820
54. Zhang, H., Weng, T.W., Chen, P.Y., Hsieh, C.J., Daniel, L.: Efficient neural network robustness certification with general activation functions. In: NIPS'18: Proceedings of the 32nd International Conference on Neural Information Processing Systems (2018), https://dl.acm.org/doi/10.5555/3327345.3327402
55. Zhang, X., Wang, B., Kwiatkowska, M.: Provable preimage under-approximation for neural networks. In: Tools and Algorithms for the Construction and Analysis of Systems (2024). https://doi.org/10.1007/978-3-031-57256-2_1