Talking with Verifiers: Automatic Specification Generation for Neural Network Verification

Yizhak Yisrael Elboher[1], Reuven Peleg[1], Zhouxing Shi[2], Guy Katz[1], Jan Křetínský[3]
[1]Hebrew University of Jerusalem, {yizhak.elboher, reuven.peleg, g.katz}@mail.huji.ac.il
[2]University of California, Riverside, zhouxing.shi@ucr.edu
[3]Masaryk University, Brno, Czech Republic, jan.kretinsky@tum.de

***Abstract*— Neural network verification tools currently support only a narrow class of specifications, typically expressed as low-level constraints over raw inputs and outputs. This limitation significantly hinders their adoption and practical applicability across diverse application domains where correctness requirements are naturally expressed at a higher semantic level. This challenge is rooted in the inherent nature of deep neural networks, which learn internal representations that lack an explicit mapping to human-understandable features. We begin addressing this gap by introducing a novel component to the verification pipeline, making existing verification tools applicable to a broader range of domains and specification styles. Our framework enables users to formulate specifications in natural language, which are then automatically analyzed and translated into formal verification queries compatible with state-of-the-art neural network verifiers. The verifier's formal guarantee applies to the generated low-level specification after the parsed intent and grounding have been approved. We evaluate our approach on both structured (tabular) and unstructured (image) datasets, demonstrating that it successfully verifies complex semantic specifications that were previously much less accessible to laymen. Our results show that this translation process maintains high fidelity to user intent while incurring low computational overhead, thereby substantially extending the applicability of formal neural network verification to real-world, high-level requirements.**

I. Introduction

Neural networks (NNs) are widely deployed in safety-critical systems, including autonomous driving, medical diagnosis, and industrial monitoring [1]–[5]. This deployment has motivated substantial progress in formal verification techniques for neural networks, which provide mathematical guarantees that a trained model satisfies a given correctness specification [6]–[9]. Existing verification tools can handle a broad class of numerical specifications, including local robustness to norm-bounded perturbations, geometric transformations, and structured input constraints [10]–[13].

However, current verification frameworks require specifications to be expressed as low-level numerical constraints, typically over fixed input dimensions. This limits both expressiveness and usability: many semantically meaningful requirements cannot be naturally encoded, and end-users must translate high-level intent into specialized specification languages. As a result, a substantial class of practically relevant specifications remains beyond the reach of existing verification pipelines.

Motivating examples. We illustrate the gap with two concrete cases.

1) *Structured features.* Consider the following scenario. A loan applicant who was denied asks why his friend, who earns 100 euros less, was approved. The bank is required to provide an explanation, but the applicant does not know how to formally express the query. A bank clerk may identify the intended property — that any applicant with sufficiently lower income, while all other features remain fixed, should also be rejected — yet writing such a specification directly in VNNLIB [14], [15] is cumbersome.
   For example, the applicant's natural-language query may be:
   "Would I still be rejected if my income were lower?"
   Assuming that the query is denoted x and that its third coordinate, x_3, represents income, this query can be encoded as
   $$\exists x'. \qquad x'_3 < x_3 \qquad \wedge \qquad \bigwedge_{j \neq 3} x'_j = x_j$$
   together with an output constraint requiring the model's decision on x' to change from rejection to approval.
   Our pipeline can automatically analyze and formalize the request, reformulate it in precise natural language for human validation, and verify the resulting property. The clerk supervises and validates the intermediate results, as required legally and practically. This workflow is complementary to feature-attribution and counterfactual-explanation methods: those methods estimate influential features or search for alternative inputs that change a decision, whereas verification can either prove that no counterexample exists within an approved perturbation region or return a concrete witness. The resulting SAFE/UNSAFE guarantee is conditional on approval of the parsed feature and grounded low-level specification.
2) *Unstructured inputs.* Consider an image classifier, an input image containing a bird, and the specification:
   "This bird image is classified correctly even if its beak is occluded."
   Here, the relevant input region — the beak — varies across input images, and cannot be captured by a fixed set

https://doi.org/10.34727/2026/isbn.978-3-85448-093-8_68

This article is licensed under a Creative Commons Attribution 4.0 International License

of input coordinates. Consequently, the property is cumbersome to express directly using uniform perturbation regions (e.g., ℓ_p-balls or global brightness changes). The current prototype generates this property separately for each concrete image; it does not prove the corresponding global statement for all bird images.

Goal. Our objective is to translate natural-language semantic specifications into standard numerical verification queries compatible with existing NN verifiers. Concretely, given a network N, a natural-language specification P, and a concrete input x, we construct a grounded specification P_x. Once the parsed intent and grounding are approved, standard tools can decide whether $N \models P_x$ or produce a counterexample. This `SAFE`/`UNSAFE` result applies to P_x, rather than unconditionally to the original natural-language sentence.

Technique overview. Our approach is built around a simple but effective insight: while current NN verifiers are powerful, their practical applicability is limited by the difficulty of writing low-level specifications. Rather than developing new verification algorithms, we focus on an integration layer that connects recent advances in foundation models and perception systems with existing neural network verification engines. Concretely, we assemble a lightweight automated pipeline in which a large language model interprets a natural-language specification, an off-the-shelf perception model grounds the referenced semantic objects in a concrete input, and a specification generator translates this grounding into a standard numerical verification query. The verifier itself remains unchanged.

This design deliberately emphasizes reusing mature components instead of introducing complex new machinery: each stage relies on readily available tools, and improvements in language or perception models can improve specification grounding. Our contribution is therefore primarily an application and integration effort that makes existing formal verification technology more accessible for the supported local specifications, without modifying the underlying verification algorithms.

Evaluation overview. A comprehensive evaluation of usability would ideally involve a large-scale user study in an industrial setting, which is beyond the scope of this work and would require substantial additional resources. Instead, we provide an illustrative evaluation that demonstrates the feasibility and practical potential of our approach. We instantiate the pipeline for two representative settings: (i) tabular models with structured features, and (ii) image classifiers using open-vocabulary object detection. We show that the generated grounded specifications can be directly consumed by state-of-the-art verifiers to check local semantic robustness properties whose input-dependent regions are cumbersome to encode manually. Our experiments demonstrate that the pipeline can produce verifier-ready queries without modifying the underlying verification backends, while also exposing important limitations of visual grounding.

Contributions. Our contribution is an integration layer between high-level semantic property specification and existing neural network verification engines. All components of the pipeline use existing models or tools; no modification to the verifier is required. This allows the approach to be used with current verification systems and supports evaluation of local semantic robustness specifications with input-dependent regions. This application framework connects foundation models with NN verification and emphasizes practical impact and reuse of established theoretical verification machinery. Concretely, our contributions are:

1) **Motivating a usability-driven specification gap.** We highlight a growing practical and societal need for specifying high-level, semantically meaningful correctness requirements for deployed NN systems, and identify the current specification bottleneck as a key obstacle to wider adoption of formal verification. In particular, we emphasize the substantial manual effort and expertise required to translate user intent into low-level verifier-specific constraints, a recurring pain point for formal methods practitioners and end-users alike.
2) **An automated specification generation mechanism.** We present a mechanism that maps user intent expressed in natural language to local formal verification specifications by identifying relevant input features or semantic objects and constructing corresponding constraints. The current prototype accepts one operation from a fixed supported set and grounds each property in a concrete input. We outline how the mechanism can be instantiated across structured (tabular) and unstructured (image) input domains.
3) **Empirical demonstration of applicability.** We conduct experiments on benchmarks of two modalities, showing that the mechanism can produce verifier-ready specifications and quantifying failures in the parsing and visual-grounding stages.

Together, these contributions provide a practical interface for constructing supported local verification specifications from natural language. While the individual technical components rely on existing models and verifiers, their composition offers a useful workflow for reducing manual feature and coordinate engineering. We view this work as a first step toward broader adoption of high-level specification-driven verification. A comprehensive usability evaluation measuring whether users find the generated specifications aligned with their intent is important future work; here, we focus on demonstrating the feasibility of the proposed integration.

II. Preliminaries

A. Formal Verification of specifications in NNs

Given a neural network $\mathcal{N} : \mathbb{R}^n \to \mathbb{R}^m$ that maps each input vector $x \in \mathbb{R}^n$ to $y = \mathcal{N}(x) \in \mathbb{R}^m$, an input specification $\psi_{in}(x)$, and an undesired output specification $\psi_{out}(y)$, the formal verification of the local specification can be formulated as $P := \psi_{in}(x) \wedge \psi_{out}(y)$, where `SAFE` is returned if P is

unsatisfiable, i.e. there does not exist $x \in \mathbb{R}^n$ for which both $\psi_{in}(x)$ and $\psi_{out}(y)$ holds, and UNSAFE is returned otherwise.

B. Language and Vision Grounding

Large Language Models (LLMs) serve as semantic interfaces, inferring user intent to produce machine-interpretable representations. Vision-Language Models (VLMs) extend this capability by jointly processing textual and sensory inputs, enabling semantic recognition through natural-language queries. In our framework, these models act as semantic translators: LLMs structure the specifications, while VLMs ground them in specific input samples to construct constraints compatible with formal NN verifiers.

C. Zero-Shot and Open-Vocabulary Reasoning

Zero-shot learning (ZSL) allows models to reason about novel concepts by leveraging semantic embeddings rather than task-specific retraining [16]–[18]. *Open-vocabulary learning* expands this by removing fixed label restrictions, allowing arbitrary user-defined concepts to be processed at inference time [19]. Models like CLIP [20] align diverse modalities in a shared space, enabling the translation of unrestricted user intents into grounded, context-dependent specifications.

D. Open-Vocabulary Detection and Localization

Beyond global representations, localized grounding is achieved via open-vocabulary object detection. In the visual domain, models such as GroundingDINO (GDINO) [21] and Grounded SAM (GSAM) [22] localize specific objects described in text. We leverage these models to tie verification specifications to precise spatial regions within the input.

III. Verification with Context-Grounded Semantic Specifications

Problem setting. A standard verification query consists of a neural network N and a formal specification P, typically expressed as input and output constraints. While existing specification languages are expressive, they are often hard to instantiate in practice, since the specification must be defined per input. This makes it difficult to capture context-dependent semantic properties that depend on the input content.

Consider the specification: *"The bird in the image is classified correctly even if its beak is occluded."* This statement cannot be directly encoded without first selecting image-specific pixels, since the location of the beak varies across inputs. Our goal is to transform such a specification into a conventional local verification specification P_x for each concrete input x; the current instantiation does not establish a global property over all inputs.

Formally, given a natural-language property P and an input x, we compute a grounded specification P_x such that standard verification tools can confirm that $N \models P_x$, or return a counterexample.

Grounding pipeline. Alg. 1 summarizes the pipeline, which is also visualized in Fig. 4. The process consists of three stages. (1) *Parsing.* A parser extracts from P: (i) a set of semantic objects to be located in the input, and (ii) one operation, selected from the fixed set supported by the current prototype, describing how these objects should be modified. (2) *Detection.* A detector localizes the parsed objects in the concrete input x, returning coordinates or time intervals. (3) *Specification generation.* A grounded formal specification P_x is constructed from x, the detected coordinates, and the parsed operation. Lines 3–5 implement a validation stage in which the user reviews and approves the outputs of the parsing and detection stages. If approval is not granted, the algorithm returns UNKNOWN.

Algorithm 1 Semantic specification grounding

Require: Network N, natural-language property P, input x
Ensure: Verification result SAFE, UNSAFE, or UNKNOWN
1: $(objects, operation) \leftarrow \textsc{Parser}(P)$
2: $coords \leftarrow \textsc{Detector}(x, objects)$
3: approved $\leftarrow \textsc{UserPreview}(objects, operation, coords)$
4: **if** ¬approved **then**
5: **return** UNKNOWN ▷ Or refine the grounding and repeat
6: **end if**
7: $P_x \leftarrow \textsc{SpecGenerator}(x, coords, operation)$
8: **return** $\textsc{Verifier}(N, P_x)$

Example instantiation. For the running example, the parser (Alg. 1, Line 1) extracts:

$$objects = \{\text{“bird beak”}\}, \quad operation = \text{“occlude”}.$$

The detector (Alg. 1, Line 2) returns pixel coordinates $coords$ of the beak in image x. The specification generator (Alg. 1, Line 6) produces the formal perturbation constraint:

$$P_x : \forall x' \in \mathcal{B}(x, coords),\ N(x') \neq N(x),$$

where $\mathcal{B}(x, coords)$ denotes all images obtained by masking the detected beak region. This is a standard local robustness property, directly consumable by existing verifiers (Alg. 1, Line 7). To improve alignment with the user's requirement, the pipeline presents intermediate artifacts — such as the parsed object or tabular feature, operation, and selected region — for approval (Alg. 1, Lines 3–5). The user need not inspect the VNNLIB encoding itself. A rejected grounding returns UNKNOWN in the basic algorithm, but can instead trigger refinement of the object phrase, detector, tightness mode, or manually selected region.

Conditional guarantee and failure modes. The parser and detector are not part of the certified system; they only propose the specification grounding. Consequently, a SAFE or UNSAFE result has the standard formal meaning of the underlying verifier only with respect to the approved grounded specification P_x. An UNSAFE result provides a concrete witness for that query, whereas a SAFE result does not by itself prove that the original natural-language sentence was interpreted correctly. The pipeline returns UNKNOWN when the grounding is rejected and may also return UNKNOWN

when the underlying verifier cannot resolve the query. This distinction is especially important in the image setting, where the workflow can also serve as a useful falsification tool when a counterexample is found.

Domain instantiations. We provide two concrete modular instantiations of our verification pipeline for tabular and image classification tasks using existing foundation models, as shown in Fig. 6. We discuss an additional audio-domain instantiation in App. C.

1) *Explicit features domain.* The parser (LLM) extracts referenced feature names and relational constraints from P. Since features correspond directly to input coordinates, the detection stage reduces to a fixed mapping from parsed features to indices in the tabular input. The specification generator then produces standard numerical constraints over the identified dimensions. Despite its simplicity, this instantiation supports practically relevant, legally and ethically sensitive requirements, where human oversight and auditability are essential, and automation must be both realistic and trustworthy.
2) *Image domain.* The parser is implemented using a large language model that extracts object descriptions and operations from P. The detector is an open-vocabulary zero-shot object detection [21] or segmentation [22] model, which localizes the parsed objects in the image. The output coordinates are passed to the specification generator, which produces region-based perturbation constraints. This supports input-dependent regions, such as a bird's beak, a traffic-sign region rather than its background, or a described anatomical region in a medical image. The detector is used only to propose a grounding; the formal result comes from verifying the approved query against the target classifier.

A user who can directly select the relevant feature, region, and operation may bypass the parser and detector and pass this information directly to the specification generator. Thus, our mechanism does not claim greater formal expressiveness than such a manual workflow. Its added value is twofold. First, *simplification*: users can refer to semantic concepts without knowing feature indices, data encodings, exact object terminology, or pixel coordinates, while the pipeline translates these descriptions into the required low-level inputs. Second, *automation*: the same natural-language property can be applied across multiple inputs, with the relevant coordinates grounded separately for each input instead of being selected manually whenever an object's location changes. These benefits introduce reliance on the parser and detector, which is why user approval or manual correction remains necessary.

IV. Evaluation

We evaluate our mechanism on two representative benchmarks covering both structured and perceptual domains:

1) A fully-connected network on Statlog (German Credit Data), a tabular dataset mapping financial and personal customer attributes to a credit risk classification.
2) A ResNet-18 classifier on the CUB-200-2011 dataset for fine-grained bird species classification.

Dataset characteristics are summarized in App. B and Tab. III.

Experimental setup. For parsing, we employ Gemini 3 Flash and GPT 5 Mini via API operating in two modes: *detailed*, for refined object representations, and *minimal*, for concise output. Experiment-specific system prompts are provided in App. E. For visual grounding, we utilize either GDINO or GSAM. We define two configurations based on confidence and relevance thresholds: *tight* (higher thresholds, all detections retained) and *loose* (lower thresholds, redundant bounding boxes removed). The specific thresholds can be found in App. D. In *loose* mode, we eliminate boxes that fully contain others to isolate the most granular semantic regions. For Tab. II, we manually reviewed 40 images and their associated queries and labeled the grounding as accepted or rejected according to whether the detected region was adequate for generating the intended verification query. We use this binary criterion because the pipeline must either proceed with the region or reject it and return `UNKNOWN`. Additional acceptance and rejection criteria, a broader set of test cases, and evaluations involving multiple annotators would provide a more comprehensive assessment and are left for future work.

For the tabular domain, we use the Statlog German Credit dataset [23] (see App. G for additional details). We generate 40 natural-language queries and translate each into a VNNLIB [14], [15] specification using our framework. For each query, we choose an input sample as follows: we consider all training samples whose model prediction is consistent with the query. For example, for the query above *"Would I still be rejected if my income were lower?"*, we consider all samples whose predicted outcome is *rejected*. Among all such samples, we choose the one whose output is closest to the decision boundary (nearest to the 0.5 threshold), specifically to stress-test difficult local verification queries rather than to estimate generalization performance. The verification result is defined per input and property, so using a training sample does not change its formal meaning; correctly classified validation or test samples could be used instead. Each specification perturbs a single attribute by $\pm\delta$ while pinning all other attributes to their original values; we sweep over 15 values of δ ranging from 10^{-4} to 50, yielding $40 \times 15 = 600$ verification instances in total.

Formal verification is performed with α-β-CROWN [9] with a timeout of 60 seconds per query.

Quantitative results. Our evaluation indicates that the semantic parsing phase performs well on the tested prompts and benchmarks. As shown in Tab. I, both GPT 5 Mini and Gemini 3 Flash achieve parsing accuracies between 85% and 100% for identifying objects and intended actions. While GPT 5 Mini generally offers the highest accuracy (reaching 100% action accuracy on the Statlog dataset), Gemini 3 Flash provides a significantly faster inference time, particularly on the Statlog benchmark (1.07 ± 0.33 seconds). These results establish feasibility for this fixed task and prompt set, but

TABLE I: Model Success Rate on Parsing

Model	Dataset	Mode	Acc. (object)	Acc. (action)	Time (Sec.)
Gemini 3 Flash	CUB-200	detailed	85%	95%	1.86 ± 2.05
GPT 5 Mini	CUB-200	detailed	95%	98%	5.29 ± 3.25
Gemini 3 Flash	CUB-200	minimal	95%	98%	3.1 ± 7.52
GPT 5 Mini	CUB-200	minimal	98%	95%	5.25 ± 2.13
GPT 5 Mini	Statlog	minimal	98%	100%	5.37 ± 2.46
Gemini 3 Flash	Statlog	minimal	98%	98%	1.07 ± 0.33

TABLE II: GDINO Grounding Acceptance Rate with parser GPT 5 Mini

Dataset	Mode	Tightness	Accepted
CUB-200-2011	detailed	loose	55%
	detailed	tight	23%
	minimal	loose	55%
	minimal	tight	38%
	any	any	83%
Statlog	-	-	100%

do not imply robustness to unrestricted or ambiguous natural-language requests.

In contrast, the perceptual grounding phase on the CUB-200-2011 dataset highlights the difficulty of open-vocabulary detection. As detailed in Tab. II, the highest acceptance rate for a single configuration was 55% using the *loose* tightness mode. The *any Mode, any Tightness* row reaches 83%, meaning that at least one tested configuration localized an acceptable region. This is an oracle-style upper bound, not evidence that a fully automated pipeline can blindly select the correct mode. A practical interactive extension can instead run several configurations or detectors in parallel, present their candidate regions to the user, and proceed only when one is approved; a failed first candidate may trigger phrase editing, a different mode or detector, or manual region adjustment. For GSAM, the success rate was 50%, with an additional 26% of partial successes, such as detecting one part of a multipart object.

Limitations. The evaluation is a proof of concept rather than a usability study. Parsing or grounding errors can yield misleading queries if they are not caught during approval, and the image-grounding results show that such failures are common. Moreover, the current prototype supports only the fixed operations listed in App. E and only local, per-input properties. The visual annotations were binary judgments by one author; quantitative or multi-level labels, inter-annotator agreement, and a user study measuring alignment with user intent are important future work.

Critical perturbation range. A practical challenge in tabular verification is that users are typically unfamiliar with the numerical scale of the input domain and therefore cannot specify a meaningful perturbation magnitude δ upfront. Our framework tackles this limitation by sweeping over a range of δ values and aggregating the outcomes. For the loan example, the result identifies the range in which the loan decision changes. Fig. 1 summarizes the result: for each query we identify the *critical delta range* — the interval $[\delta_{\text{safe}}, \delta_{\text{unsafe}}]$ where δ_{safe} is the largest δ for which the model is verified

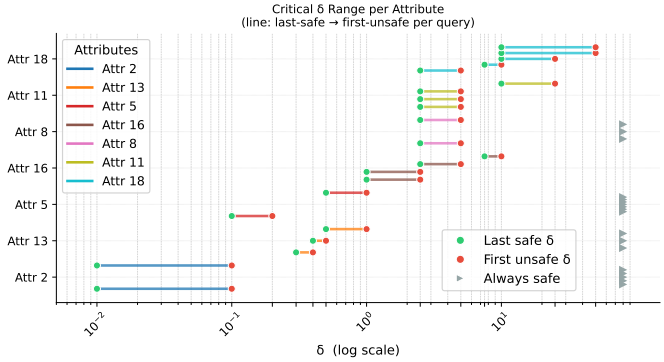


Fig. 1: Critical δ range per attribute on the Statlog German Credit dataset. Each horizontal line spans from the last δ at which the query is verified `SAFE` (green dot) to the first δ at which a counterexample is found (`UNSAFE`, red dot). Multiple lines per attribute correspond to individual queries. Grey markers (▷) denote queries that remain `SAFE` across all tested δ values.

robust and δ_{unsafe} is the smallest δ at which a counterexample is found. This range pinpoints how robust the model is with respect to the queried attribute, without requiring any domain knowledge from the user. Attributes such as *Duration* (Attr 2) and *Age* (Attr 13) become unsafe already at small perturbations ($\delta \approx 0.1$–0.4), indicating that the model's decision is sensitive to small changes in these features. In contrast, *Installment rate* (Attr 8) and *Number of dependents* (Attr 18) remain robust until much larger perturbations ($\delta \geq 5$). Queries that never become `UNSAFE` across the entire sweep are shown as grey markers beyond the x-axis range, meaning the model is provably robust to any perturbation magnitude tested.

Additional results on Statlog can be found at App. F.

Qualitative results. We qualitatively evaluate the pipeline's ability to translate high-level user intent into precise verification constraints, using the visual domain as a primary case study. Fig. 2 illustrates the end-to-end execution of a semantic query on the CUB-200-2011 dataset.

The process initiates with the natural language specification: "Can the prediction change if all the purple thorns in the image are partially occluded?" (Fig. 2a). This request presents a challenge for traditional verifiers because it relies on semantic concepts ("purple thorns") and a global aggregation over multiple instances, rather than fixed numerical input dimensions. The pipeline handles this by first employing the LLM-based parser to decompose the user's intent. The parser correctly identifies the target object and the requested operation, formulating the grounding query "purple thorns" (Fig. 2b). This intermediate representation is critical, as it strips away the verificational context ("Can the prediction change…") to focus solely on the perceptual task.

Next, the open-vocabulary detector (GDINO) processes this query alongside the original image (Fig. 2d). As demonstrated in (Fig. 2e), the model successfully identifies multiple relevant instances, placing bounding boxes around all purple thorns in the image. This comprehensive localization enables the

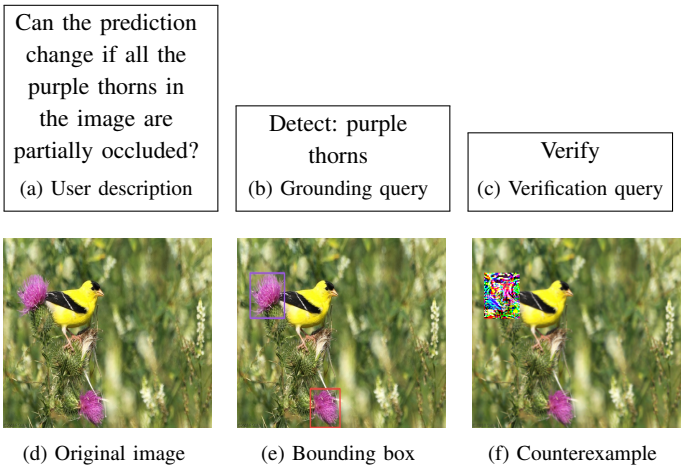


Fig. 2: Qualitative demonstration of specification grounding in the full verification process. The user description (a) is parsed into a grounding query (b). The semantic detector applies this query to the image (d) to identify the relevant semantic region (e) and generate a counterexample (f).

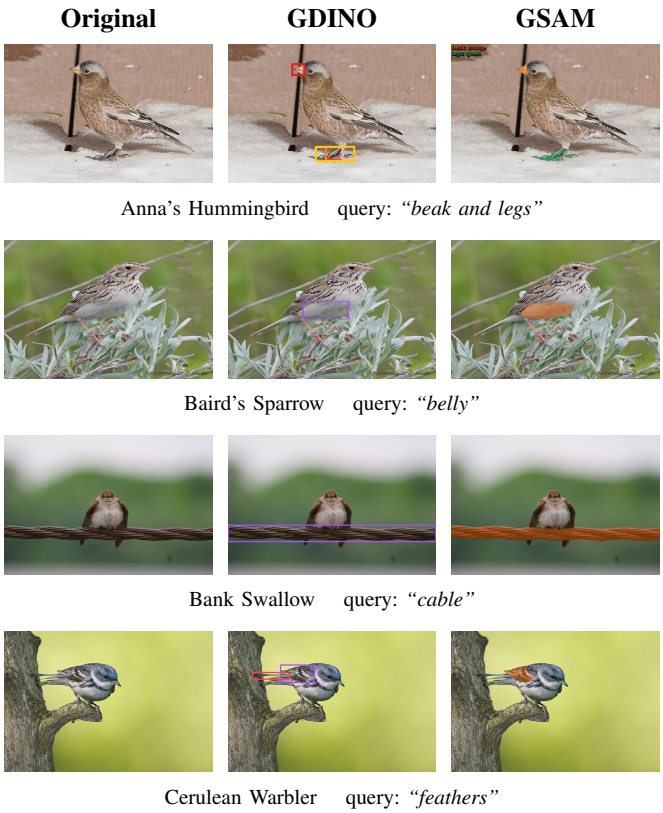


Fig. 3: Qualitative comparison of GDINO and GSAM on four CUB images. Each row shows the original image (left), the GDino bounding-box detection (center), and the GSAM pixel-level segmentation mask (right) for the indicated query object. GDino marks a rectangular region around the target; GSAM refines it to a precise (orange/green colored) mask that follows the object boundary and excludes surrounding background.

Specification Generator to define a grounded specification P_x. Instead of a global perturbation that might affect the entire input, the generator constructs a set of local robustness constraints restricted to the pixels within the detected regions, modeling partial occlusion. This effectively isolates all semantic regions of interest, allowing the underlying verifier to check the model's robustness against occlusion of the "purple thorns" without manual coordinate engineering.

The illustration in Fig. 3 qualitatively shows (i) different ways to detect an object with either an object detection model (GDINO) or a segmentation model (GSAM), and (ii) that the mechanism can detect multiple targets from a user description, whether they are distinct objects ("beak and legs") or multiple parts of a single object ("feathers").

V. Related Work

This work tries to increase the adoption of neural network verification. While other directions are to improve the scalability of the verification process [24]–[31], to extend the range of architectures that can be verified [32]–[36], and to apply formal verification of neural networks in multiple domains and tasks [37]–[39], we focus on extending the expressivity range of specifications that can be verified.

Typically, specifications are categorized as either global or local. Global specifications [40]–[42] provide input-agnostic guarantees but are computationally difficult. Local specifications concern correctness in a neighborhood of a specific sample and are more tractable. We focus on local specifications, as our framework requires concrete samples to ground natural-language intents into formal constraints.

Foundation models have recently been used to assist in verification tasks. In program verification, LLMs guide search heuristics, generate invariants, or synthesize specifications [43], [44]. For NNs, LLMs assist in property specification and debugging [45], [46], while VLMs provide semantic annotations for robustness and interpretability [47]. In the loan example, our workflow is complementary to feature-attribution, sensitivity, and counterfactual analyses: rather than only identifying influential features or searching for an alternative decision, it asks the verifier to prove or falsify an approved local perturbation property.

Despite efforts to standardize specifications [14], [15], no prior work systematically uses multimodal models to broaden the expressiveness of specifications by translating open-vocabulary user intents into grounded formal queries. Our approach fills this gap by bridging semantic interpretation with standard verification backends.

VI. Conclusion & Future Work

We presented a proof-of-concept framework for translating supported natural-language requests into local neural-network verification specifications. Its primary value lies in the modular integration of parsers, grounding models, specification generators, and existing verifiers. The framework preserves the soundness of the underlying verifier with respect to an approved grounded specification; it does not certify that an unapproved or incorrectly grounded query matches the original

natural-language intent. Accordingly, rejected groundings and unresolved verification queries are reported as `UNKNOWN`.

Future work includes extending the framework to video [48], supporting temporal constraints for audio events (see App. C), improving prompts and grounding models, adding parallel candidate selection and iterative refinement, using multi-level and multi-annotator grounding assessments, and conducting a broader usability study of specification alignment.

VII. Acknowledgements

The work of Elboher, Peleg, Katz and Křetínský was partially funded by the European Union (RobustifAI project, ID 101212818). Views and opinions expressed are however those of the author(s) only and do not necessarily reflect those of the European Union or the European Health and Digital Executive Agency (HADEA). Neither the European Union nor the granting authority can be held responsible for them.

References

[1] O. Shobayo and R. Saatchi, "Developments in Deep Learning Artificial Neural Network Techniques for Medical Image Analysis and Interpretation," *Diagnostics*, 2025.

[2] J. Zhao, Y. Wu, R. Deng, S. Xu, J. Gao, and A. Burke, "A Survey of Autonomous Driving from a Deep Learning Perspective," *ACM Computing Surveys (CSUR)*, 2025.

[3] Z. Salahuddin, H. C. Woodruff, A. Chatterjee, and P. Lambin, "Transparency of Deep Neural Networks for Medical Image Analysis: A Review of Interpretability Methods," *Computers in Biology and Medicine*, 2022.

[4] R. Khanam, M. Hussain, R. Hill, and P. Allen, "A Comprehensive Review of Convolutional Neural Networks for Defect Detection in Industrial Applications," *IEEE Access*, pp. 94 250–94 295, 2024.

[5] Z. Li, Q. He, and J. Li, "A Survey of Deep Learning-Driven Architecture for Predictive Maintenance," *Int. Journal Engineering Applications of Artificial Intelligence (IJEAAI)*, 2024.

[6] R. Ehlers, "Formal Verification of Piece-Wise Linear Feed-Forward Neural Networks," in *Proc. of the Conf. on Automated Technology for Verification and Analysis (ATVA)*, 2017.

[7] G. Katz, C. Barrett, D. L. Dill, K. Julian, and M. J. Kochenderfer, "Reluplex: An efficient SMT Solver for Verifying Deep Neural Networks," in *Proc. 29th Int. Conf. Computer Aided Verification (CAV)*, 2017, pp. 97–117.

[8] V. Tjeng, K. Y. Xiao, and R. Tedrake, "Evaluating Robustness of Neural Networks with Mixed Integer Programming," in *Proc. 7th Int. Conf. on Learning Representations (ICLR)*, 2019.

[9] S. Wang, H. Zhang, K. Xu, X. Lin, S. Jana, C.-J. Hsieh, and J. Z. Kolter, "Beta-crown: Efficient Bound Propagation with Per-Neuron Split Constraints for Neural Network Robustness Verification," *Proc. 34th Conf. on Advances in Neural Information Processing Systems (NeurIPS)*, pp. 29 909–29 921, 2021.

[10] M. Balunovic, M. Baader, G. Singh, T. Gehr, and M. Vechev, "Certifying Geometric Robustness of Neural Networks," in *Proc. 32nd of Conf. on Advances in Neural Information Processing Systems (NeurIPS)*, 2019.

[11] J. Mohapatra, T.-W. Weng, P.-Y. Chen, S. Liu, and L. Daniel, "Towards Verifying Robustness of Neural Networks Against A Family of Semantic Perturbations," in *Proc. IEEE/CVF Conf. on Computer Vision and Pattern Recognition (CVPR)*, 2020, pp. 241–249.

[12] B. Batten, Y. Zheng, A. D. Palma, P. Kouvaros, and A. Lomuscio, "Verification of Geometric Robustness of Neural Networks via Piecewise Linear Approximation and Lipschitz Optimisation," 2024, Technical Report. https://arxiv.org/abs/2408.13140.

[13] Y. Elboher, R. Elsaleh, O. Isac, M. Ducoffe, A. Galametz, G. Povéda, R. Boumazouza, N. Cohen, and G. Katz, "Robustness Assessment of a Runway Object Classifier for Safe Aircraft Taxiing," in *Proc. 43rd AIAA DATC/IEEE Digital Avionics Systems Conf. (DASC)*, 2024, pp. 1–6.

[14] S. Demarchi, D. Guidotti, L. Pulina, and A. Tacchella, "Supporting Standardization of Neural Networks Verification with VNNLIB and CoCoNet," in *Proc. 6th Workshop on Formal Methods for ML-Enabled Autonomous Systems (FoMLAS)*, 2023, pp. 47–58.

[15] A. Roy, A. Antony, A. Gimelli, and M. L. Daggitt, "Vnn-lib 2.0: Rigorous foundations for neural network verification," 2026. [Online]. Available: https://arxiv.org/abs/2605.07451

[16] R. Socher, M. Ganjoo, C. D. Manning, and A. Ng, "Zero-Shot Learning Through Cross-Modal Transfer," in *Proc. 26th Int. Conf. Advances in Neural Information Processing Systems (NeurIPS)*, 2013.

[17] Y. Xian, B. Schiele, and Z. Akata, "Zero-Shot Learning — The Good, the Bad and the Ugly," in *Proc. IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)*, 2017, pp. 3077–3086.

[18] T. Kojima, S. S. Gu, M. Reid, Y. Matsuo, and Y. Iwasawa, "Large language models are zero-shot reasoners," in *Proc. 36th Int. Conf. on Neural Information Processing Systems (NeurIPS)*, 2022.

[19] J. Wu, X. Li, S. Xu, H. Yuan, H. Ding, Y. Yang, X. Li, J. Zhang, Y. Tong, X. Jiang, B. Ghanem, and D. Tao, "Towards Open Vocabulary Learning: A Survey," *Proc. IEEE Transactions on Pattern Analysis and Machine Intelligence (TPAMI)*, pp. 5092–5113, 2024.

[20] A. Radford, J. W. Kim, C. Hallacy, A. Ramesh, G. Goh, S. Agarwal, G. Sastry, A. Askell, P. Mishkin, J. Clark, G. Krueger, and I. Sutskever, "Learning Transferable Visual Models From Natural Language Supervision," in *Proc. 38th of Int. Conf. on Machine Learning (ICML)*, 2021.

[21] S. Liu, Z. Zeng, T. Ren, F. Li, H. Zhang, J. Yang, Q. Jiang, C. Li, J. Yang, H. Su, J. Zhu, and L. Zhang, "Grounding DINO: Marrying DINO with Grounded Pre-training for Open-Set Object Detection," in *Proc. of European Conf. on Computer Vision (ECCV)*, 2025, pp. 38–55.

[22] T. Ren, S. Liu, A. Zeng, J. Lin, K. Li, H. Cao, J. Chen, X. Huang, Y. Chen, F. Yan, Z. Zeng, H. Zhang, F. Li, J. Yang, H. Li, Q. Jiang, and L. Zhang, "Grounded SAM: Assembling Open-World Models for Diverse Visual Tasks," 2024, Technical Report. https://arxiv.org/abs/2401.14159.

[23] H. Hofmann, "Statlog (German Credit Data)," UCI Machine Learning Repository, 1994, DOI: https://doi.org/10.24432/C5NC77.

[24] S. Wang, K. Pei, J. Whitehouse, J. Yang, and S. Jana, "Efficient Formal Safety Analysis of Neural Networks," *Proc. 31st Int. Conf. on Advances in Neural Information Processing Systems (NeurIPS)*, 2018.

[25] T. Gehr, M. Mirman, D. Drachsler-Cohen, P. Tsankov, S. Chaudhuri, and M. T. Vechev, "AI2: Safety and Robustness Certification of Neural Networks with Abstract Interpretation," *Proc. 39th IEEE Symposium on Security and Privacy (S&P)*, pp. 3–18, 2018.

[26] Y. Y. Elboher, J. Gottschlich, and G. Katz, "An Abstraction-Based Framework for Neural Network Verification," in *Proc. 32nd Int. Conf. Computer Aided Verification (CAV)*, 2020, pp. 43–65.

[27] P. Ashok, V. Hashemi, J. Křetínský, and S. Mohr, "DeepAbstract: Neural Network Abstraction for Accelerating Verification," in *Proc. 18th Int. Symposium on Automated Technology for Verification and Analysis (ATVA)*, 2020, pp. 92–107.

[28] Y. Y. Elboher, E. Cohen, and G. Katz, "Neural Network Verification Using Residual Reasoning," in *Proc. 20th Conf. on Software Engineering and Formal Methods (SEFM)*, 2022, pp. 173–189.

[29] E. Cohen, Y. Y. Elboher, C. Barrett, and G. Katz, "Tighter Abstract Queries in Neural Network Verification," in *Proc. 24th Int. Conf. on Logic for Programming, Artificial Intelligence and Reasoning (LPAR)*, 2023, pp. 124–143.

[30] K. Xu, H. Zhang, S. Wang, Y. Wang, S. Jana, X. Lin, and C.-J. Hsieh, "Fast and Complete: Enabling Complete Neural Network Verification with Rapid and Massively Parallel Incomplete Verifiers," in *Proc. 9th Int. Conf. on Learning Representations (ICLR)*, 2021.

[31] D. Zhou, C. Brix, G. A. Hanasusanto, and H. Zhang, "Scalable Neural Network Verification with Branch-and-bound Inferred Cutting Planes," in *Proc. 37th Conf. on Advances in Neural Information Processing Systems (NeurIPS)*, 2024, pp. 29 324–29 353.

[32] I. Khmelnitsky, D. Neider, R. Roy, B. Barbot, B. Bollig, A. Finkel, S. Haddad, M. Leucker, and L. Ye, "Property-Directed Verification of Recurrent Neural Networks," 2020, Technical Report. https://arxiv.org/abs/2009.10610.

[33] Z. Shi, H. Zhang, K.-W. Chang, M. Huang, and C.-J. Hsieh, "Robustness Verification for Transformers," 2020, Technical Report. https://arxiv.org/abs/2002.06622.

[34] Y. Y. Elboher, A. Raviv, Y. L. Weiss, O. Cohen, R. Assa, G. Katz, and H. Kugler, "Formal Verification of Deep Neural Networks for Object Detection," 2024, Technical Report. https://arxiv.org/abs/2407.01295.

[35] Y. Y. Elboher, A. Raviv, A. Elboher, Z. Shi, O. Azencot, H. Kugler, and G. Katz, "Bridging Efficiency and Safety: Formal Verification of Neural Networks with Early Exits," 2025, Technical Report. https://arxiv.org/abs/2512.20755.

[36] T. Ladner, M. Eichelbeck, and M. Althoff, "Formal Verification of Graph Convolutional Networks with Uncertain Node Features and Uncertain Graph Structure," *Transactions on Machine Learning Research (TMLR)*, 2025.

[37] G. Amir, M. Schapira, and G. Katz, "Towards Scalable Verification of Deep Reinforcement Learning," 2021, Technical Report. https://arxiv.org/abs/2105.11931.

[38] S. Bassan, Y. Y. Elboher, T. Ladner, M. Althoff, and G. Katz, "Explaining, Fast and Slow: Abstraction and Refinement of Provable Explanations," in *Proc. 42nd Int. Conf. on Machine Learning (ICML)*, 2025, pp. 3202–3225.

[39] Y. Y. Elboher, O. Isac, G. Katz, T. Ladner, and H. Wu, "Abstraction-Based Proof Production in Formal Verification of Neural Networks (Extended Abstract)," in *Proc. 2nd Int. Symposium AI Verification SAIV*, 2025, pp. 203–220.

[40] W. Ruan, M. Wu, Y. Sun, X. Huang, D. Kroening, and M. Kwiatkowska, "Global Robustness Evaluation of Deep Neural Networks with Provable Guarantees for the Hamming Distance," in *Proc. 28th Int. Joint Conf. on Artificial Intelligence (IJCAI)*, 2019, pp. 5944–5952.

[41] Z. Wang, C. Huang, and Q. Zhu, "Efficient Global Robustness Certification of Neural Networks via Interleaving Twin-Network Encoding (Extended Abstract)," in *Proc. 32nd Int. Joint Conf. on Artificial Intelligence (IJCAI)*, 2023, pp. 6498–6503.

[42] A. Kabaha and D. D. Cohen, "Verification of neural networks' global robustness," *Proc. ACM Program. Lang. (POPL)*, 2024.

[43] A. Kalyan, A. Mohta, O. Polozov, D. Batra, P. Jain, and S. Gulwani, "Neural-Guided Deductive Search for Real-Time Program Synthesis from Examples," 2018, Technical Report. https://arxiv.org/abs/1804.01186.

[44] M. Chen, J. Tworek, H. Jun, Q. Yuan, H. P. de Oliveira Pinto, J. Kaplan, H. Edwards, Y. Burda, N. Joseph, G. Brockman, A. Ray, R. Puri, G. Krueger, M. Petrov, H. Khlaaf, G. Sastry, P. Mishkin, B. Chan, S. Gray, N. Ryder, M. Pavlov, A. Power, L. Kaiser, M. Bavarian, C. Winter, P. Tillet, F. P. Such, D. Cummings, M. Plappert, F. Chantzis, E. Barnes, A. Herbert-Voss, W. H. Guss, A. Nichol, A. Paino, N. Tezak, J. Tang, I. Babuschkin, S. Balaji, S. Jain, W. Saunders, C. Hesse, A. N. Carr, J. Leike, J. Achiam, V. Misra, E. Morikawa, A. Radford, M. Knight, M. Brundage, M. Murati, K. Mayer, P. Welinder, B. McGrew, D. Amodei, S. McCandlish, I. Sutskever, and W. Zaremba, "Evaluating Large Language Models Trained on Code," 2021, Technical Report. https://arxiv.org/abs/2107.03374.

[45] M. Cosler, C. Hahn, D. Mendoza, F. Schmitt, and C. Trippel, "nl2spec: Interactively Translating Unstructured Natural Language to Temporal Logics with Large Language Models," in *Proc. 35th Int. Conf. on Computer Aided Verification (CAV)*, 2023, pp. 383–396.

[46] G. Granberry, W. Ahrendt, and M. Johansson, "Specify What? Enhancing Neural Specification Synthesis by Symbolic Methods," 2024, Technical Report. https://arxiv.org/abs/2406.15540.

[47] Z. Li, Z. Xi-Jia, B. Altundas, L. Chen, R. Paleja, and M. Gombolay, "Towards Automated Semantic Interpretability in Reinforcement Learning via Vision-Language Models," 2025, Technical Report. https://arxiv.org/abs/2503.16724.

[48] S. Sasaki, D. M. Lopez, P. K. Robinette, and T. T. Johnson, "Robustness Verification of Video Classification Neural Networks," in *Proc. 13th Int. Conf. on Formal Methods in Software Engineering (FormaliSE)*, 2025, pp. 22–33.

[49] T. Oikarinen, S. Das, L. M. Nguyen, and T.-W. Weng, "Label-Free Concept Bottleneck Models," in *Int. Conf. on Learning Representations (ICLR)*, 2023.

[50] P. Cai, Y. Song, Q. Gu, N. Jiang, H. Song, and I. McLoughlin, "Detect Any Sound: Open-Vocabulary Sound Event Detection with Multi-Modal Queries," in *Proc. 33rd ACM Int. Conf. on Multimedia (MM)*, 2025, pp. 582–591.

[51] J. Hai, H. Wang, W. Guo, and M. Elhilali, "FlexSED: Towards Open-Vocabulary Sound Event Detection," 2025, Technical Report. https://arxiv.org/abs/2509.18606.

Appendix

The appendix provides additional visualizations and technical details that are not included in the main paper.

Appendix A
Our Mechanism Flowchart

A flowchart of our verification pipeline is shown in Fig. 4. Given a network $\mathcal{N}$, a high-level specification P, and a reference input x, a parser extracts semantic objects and an operation to apply on them from P. An object detector localizes these objects in x, producing coordinates. A specification generator builds a grounded specification P_x from these coordinates and x. Finally, a verifier checks the grounded specification on nn, returning `SAFE` or `UNSAFE`.

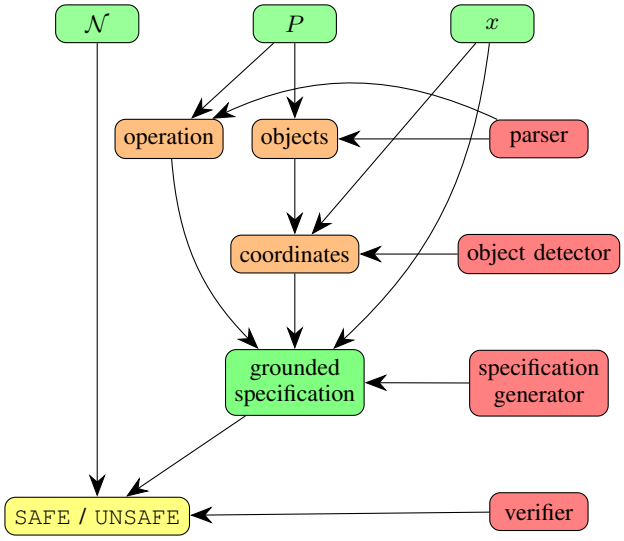


Fig. 4: Our verification pipeline.

Appendix B
Benchmarks Details

Statlog. The Statlog benchmark includes 1,000 samples with 20 features (full attribute details appear in App. G). We split the data (80/20) into train and test sets with stratification. We train a three-layer fully connected network ($20 \to 32 \to 16 \to 1$, ReLU activations, sigmoid output) using Adam with learning rate 5×10^{-3} and weight decay 10^{-4} for 300 epochs, achieving 78% test accuracy.

CUB-200-2011. The CUB-200-2011 model is taken from the publicly available implementation[1] of [49]. The model can be downloaded using the *download_models.sh* script, and the dataset can be downloaded using the *download_cub.sh* script.

Tab. III summarizes the benchmarks used in our evaluation.

Appendix C
Application in the Audio Domain.

Similarly to Vision-Language Models (VLMs), Audio-Language Models (ALMs) extend LLMs capability by jointly processing textual and sensory inputs, enabling semantic recognition through natural-language queries. In our framework, ALMs can act as semantic translators: LLMs structure the specifications, while ALMs ground them in specific input samples to construct constraints compatible with formal NN verifiers.

The instantiation of the full pipeline (Fig. 4) appears at Fig. 6b: the parser extracts semantic sound events and operations from P.

In principle, these can be grounded using an open-vocabulary sound event localization model like DASM [50] and FlexSED [51], which detect and classify sound events via natural language [51], and would return temporal intervals of relevant events. The specification generator would then produce perturbation constraints over the detected time segments.

As an illustrative example, for the property (see also Fig. 5):

"The emergency siren is detected even if the drilling noise is louder."

the parser extracts objects = {drilling noise} and operation = {amplify}. The detector identifies the time interval of drilling noise in x, and the generator produces a constraint increasing amplitude in that interval while preserving the classification. The resulting P_x is again a standard robustness query.

A. Parser Instantiation in the Audion Domain

Appendix D
Grounding DINO parameters

Grounding DINO provides thresholds on both text and box to enable control on the confidence of each. Tab. IV summarizes the configuration in each mode.

TABLE III: Datasets used in evaluation.

Dataset	Domain	#Classes	#Samples	Model	Accuracy
Statlog	Tabular	2	1000	FC (3 layers)	78.00
CUB-200-2011	Images	200	11,788	ResNet18	74.58

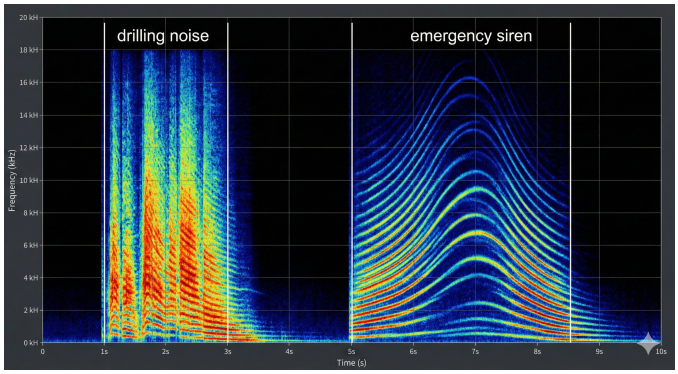
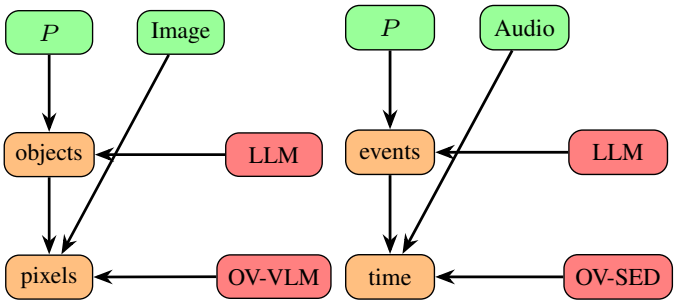


Fig. 5: Illustrative example for sound event detection. Given an audio file and an open vocabulary question to detect an event in the audio file, an open-vocabulary sound event detection (OV-SED) model predicts the start and end times of the event in the audio file. Multiple events are supported as well.

[1] https://github.com/Trustworthy-ML-Lab/Label-free-CBM



(a) Image domain (b) Audio domain

Fig. 6: Semantic grounding pipelines for different input modalities. (a) Image-based specification grounding using LLM and open-vocabulary vision-language models. (b) Sound-based specification grounding using LLM and open-vocabulary sound event detection (SED) models.

TABLE IV: Parameter Configurations for Detection Modes

Mode	Box Threshold	Text Threshold
Tight	0.35	0.25
Loose	0.15	0.15

a) Post processing in the loose mode: Since the loose mode detects more objects, we prune any bounding box that fully encloses another, assuming such cases indicate over-detection (e.g., if "beak" and "bird" are both found, and the beak is inside the bird, the bird is omitted).

APPENDIX E
SYSTEM PROMPTS

We provide the system prompts used for the parsing stage in both GPT 5 Mini and Gemini 3 Flash.

Listing 1: Visual Tasks Parsing System Prompt

```
# Role
You are a specialist in Visual
    Grounding. Your task is to
    extract the specific objects
    mentioned in a formal
    verification query that need
    to be localized in an image,
    and to identify which image
    transformation is being
    requested in each query.

# TASK
1. Analyze the user's natural
    language verification property
    .
2. Identify the objects that must
    be removed, changed, or
    checked (the "disturbing"
    objects).
3. Identify the action.
4. Pack the results into a JSON
    object with two fields: "
    object" and "action".

# SUPPORTED ACTIONS
You must categorize the request
    into exactly one of these
    supported operations:
- remove
- add_noise
- increase_brightness
- decrease_brightness
- increase_contrast
- decrease_contrast
- rotate
- scale_up
- scale_down

# RULES
{'- Use only noun phrases (e.g., "
    all cars", "the left cat").'
      if MODE == 'detailed'
      else '- Use only object
          names (e.g., "car", "cat
          ").'}
- If there are multiple distinct
    types of objects, separate
    them with a dot (e.g., "cat .
    dog").
- Do NOT include any introductory
    text, reasoning, or
    punctuation like periods at
    the end.
- Your output must be ONLY the
    result JSON.
- Produce JSON with the format
```json
{{
            "object": <object>,
            "action": <action>
}}
```

EXAMPLES
User: "Check that the
    classification of the
    pedestrian is correct even if
    the cars are not clear."
Response:
{{
            "object": "cars",
            "action": "add_noise"
}}

User: "check that the bird is
    classified correctly if both
    the beak and the tail are
    missing."
Response:
{{
            "object": "beak . tail
                ",
            "action": "remove"
}}

User: "is it possible that the car
     is misclassified when the
    brightness of its front wheels
     is increased?"
Response:
{{
            "object": {"front
```

```
                    wheels" if MODE ==
                     'detailed' else "
                    wheels"},
                "action": "
                    increase_brightness
                    "
}}
```

Listing 2: Tabular Tasks Parsing System Prompt

```
# Role
You are a specialist in Tabular
    Data Grounding. Your task is
    to extract the specific
    attributes mentioned in a user
     query and identify the
    variable and action required
    for formal verification of his
     query.

# TASK
1. Analyze the user's natural
    language verification property
    .
2. Identify the attribute that
    must be modified.
3. Identify the action.
4. Pack the results into a JSON
    object with two fields: "
    attribute" and "action".

# Attributes
Attribute2 - Duration (months)
Attribute5 - Credit amount
Attribute8 - Installment rate as a
     percentage of disposable
    income
Attribute11 - Present residence
    since
Attribute13 - Age (years)
Attribute16 - Number of existing
    credits at this bank
Attribute18 - Number of people
    liable to provide maintenance
    for

# SUPPORTED ACTIONS
You must categorize the request
    into exactly one of these
    supported operations:
- increase
- decrease
- change

# RULES
- Your output must be ONLY the
    result JSON.
- Produce JSON with the format
```json
{
      "attribute": <attribute>,
      "action": <action>
}
```

EXAMPLES
User: "Could I get the loan if I
    had fewer dependents?"
Output:
```json
{
      "attribute": "Attribute18",
      "action": "decrease"
}
```
```

APPENDIX F
STATLOG GERMAN CREDIT: EVALUATION RESULTS

We used perturbation budget $\delta \in \{10^{-4}, 10^{-3}, 10^{-2}, 0.1, 0.2, 0.3, 0.4, 0.5, 1, 2.5, 5, 7.5, 10, 25, 50\}$, yielding $40 \times 15 = 600$ verification instances in total. The results for the first three values $\{10^{-4}, 10^{-3}, , 10^{-2}\}$ were always `SAFE` so we omit the first two $\{10^{-4}, 10^{-3}\}$ which can be derived from the results for 10^{-2}, in order to save space and reduce clutter.

The figures in this section present our evaluation results on the Statlog benchmark. We report the number of `SAFE`, `UNSAFE` and `UNKNOWN` instances for each δ value (Fig. 8) and for each attribute (Fig. 9), the heatmaps of `SAFE` and `UNSAFE` results for each $\langle \delta$, attribute$\rangle$ couple (Figs. 11 and 12), the fraction of solved instances for each δ and attribute (Fig. 10), and a full image of all the solved instances results (Fig. 7).

APPENDIX G
STATLOG GERMAN CREDIT: ATTRIBUTE DETAILS

Tab. V lists all 20 input features of the Statlog German Credit dataset together with their UCI description and data type. Numeric features are treated as continuous; categorical and binary features are label-encoded to integers before training.

APPENDIX H
GROUNDINGDINO VERSUS GROUNDED SAM AS OBJECT DETECTOR

Our framework supports two spatial grounding backends that differ in the granularity of the regions they produce.

a) GroundingDINO: GroundingDINO [21] is an open-vocabulary object detector that maps a free-form text prompt to axis-aligned bounding boxes. Each box is defined by its four corner coordinates, so the "detected region" is a rectangular pixel region. Verification constraints are then expressed over all pixels inside that rectangle. GDino is fast and requires only the detection backbone.

b) Grounded SAM: Grounded SAM [22] extends GDino with the Segment Anything Model (SAM)[2]. After GDino localizes a bounding box, SAM refines it into a pixel-level segmentation mask that follows the actual object boundary. The resulting region is non-rectangular and typically tighter than the bounding box, excluding background pixels inside the box. This precision comes at the cost of running a second, heavier model (ViT-B SAM in our experiments).

[2]https://github.com/facebookresearch/segment-anything

TABLE V: Input attributes of the Statlog German Credit dataset (UCI id=144). Numeric features are treated as continuous; categorical and binary features are label-encoded.

Attr.	Description	Type
1	Status of existing checking account	Categorical
2	Duration	Numeric
3	Credit history	Categorical
4	Purpose	Categorical
5	Credit amount	Numeric
6	Savings account / bonds	Categorical
7	Present employment since	Categorical
8	Installment rate (% of income)	Numeric
9	Personal status and sex	Categorical
10	Other debtors / guarantors	Categorical
11	Present residence since	Numeric
12	Property	Categorical
13	Age (years)	Numeric
14	Other installment plans	Categorical
15	Housing	Categorical
16	Number of existing credits at bank	Numeric
17	Job	Categorical
18	Number of dependents	Numeric
19	Telephone	Binary
20	Foreign worker	Binary

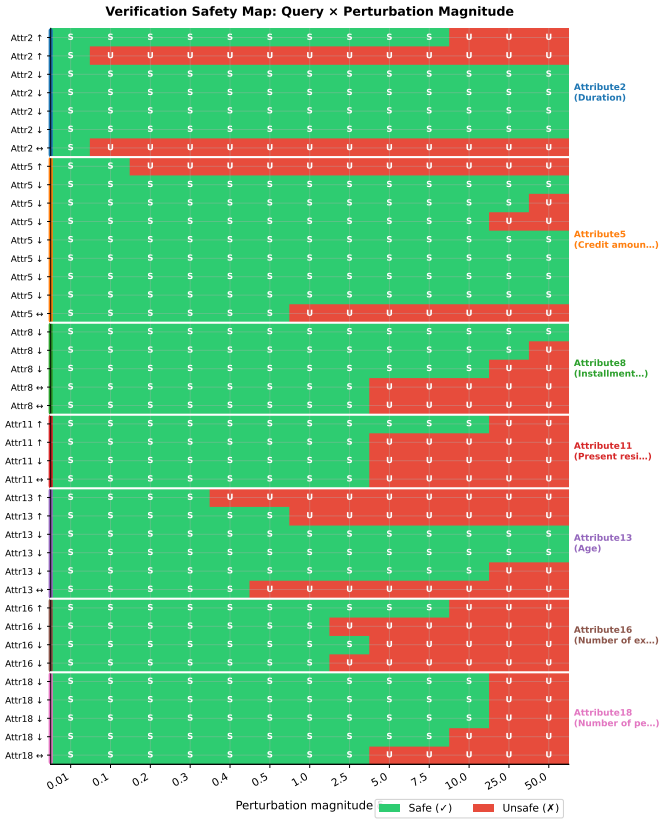


Fig. 7: **Safety map: query × δ.** Each row is one of the 40 queries (grouped by attribute; action direction shown as ↑/↓/↔); each column is a perturbation budget δ. Green (S) = the verifier proved safety; red (U) = a counterexample was found. The map reveals that small perturbations leave almost all properties safe, while specific attributes — most prominently *Duration* (Attribute 2) — become unsafe at relatively small δ.

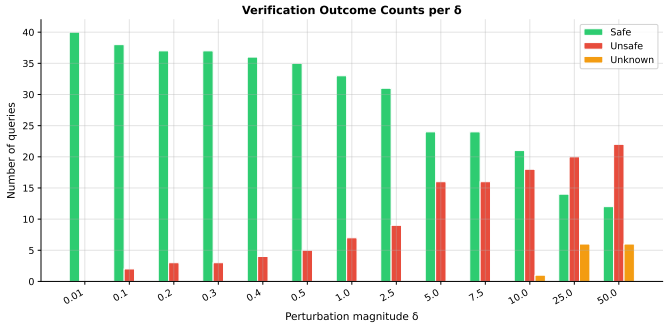


Fig. 8: **Verification outcome counts per δ.** Grouped bars show the number of safe, unsafe, and unknown results at each perturbation budget. The safe count decreases monotonically as δ grows. Unknown results (verifier timeout) appear only at $\delta \geq 10$, indicating that large perturbation budgets make some properties genuinely hard to verify within 60 s.

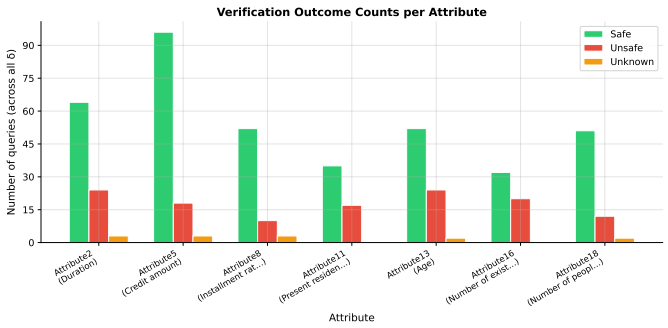


Fig. 9: **Verification outcome counts per attribute (aggregated over all δ).** Each group of bars corresponds to one attribute; bar heights show the total number of safe, unsafe, and unknown results summed across all 15 tested perturbation budgets. Attributes with more queries (*Credit amount*, Attribute 5; *Duration*, Attribute 2) naturally accumulate higher counts. The proportion of unsafe results varies considerably: *Duration* and *Age* (Attribute 13) have the highest unsafe fractions, while *Installment rate* (Attribute 8) and *Residence since* (Attribute 11) remain predominantly safe across all budgets.

A. Running time comparison.

Tab. VI reports per-query inference times over the 40 queries in our CUB evaluation (16 images, CPU only). Grounded SAM is on average *2.2× slower* than Grounding Dino, reflecting the additional SAM inference pass.

APPENDIX I
DISCLOSURE: USE OF LARGE LANGUAGE MODELS (LLMS)

The authors were solely responsible for developing the research questions, designing the methodology, performing

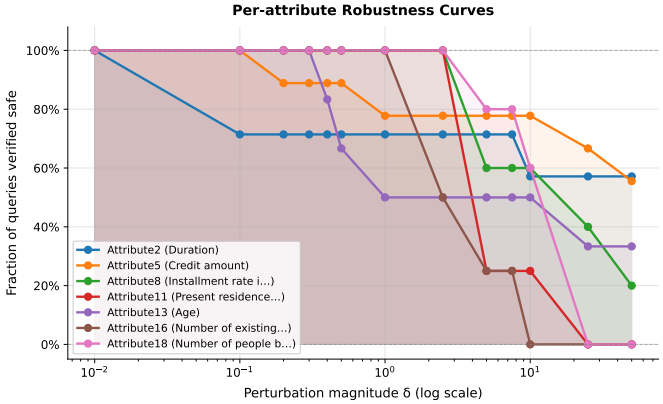


Fig. 10: **Per-attribute robustness curves.** Each curve shows the fraction of queries for one attribute that are verified safe as a function of δ (log scale, shaded area for emphasis). Attributes differ markedly in their robustness: *Duration* (Attribute 2) degrades earliest, while *Installment rate* (Attribute 8) and *Residence since* (Attribute 11) remain fully safe over a wider range.

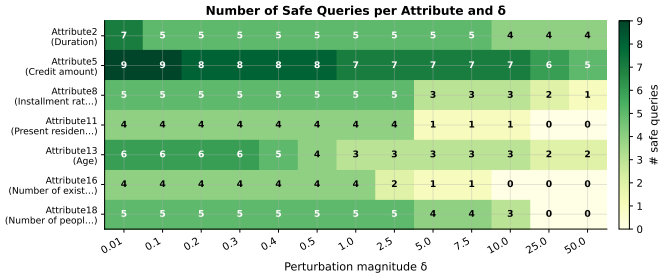


Fig. 11: **Number of safe queries per attribute and δ.** Cell values indicate how many queries for each attribute are verified safe at each δ. Lighter cells correspond to fewer safe queries. Attributes with many queries (*Credit amount*, Attribute 5) show a broader spread, while single-query attributes disappear abruptly once their critical δ is exceeded.

the analysis, and interpreting the findings. A large language model was employed only to assist with improving the clarity and style of the writing and implementing standard functions, without influencing any substantive aspects of the research.

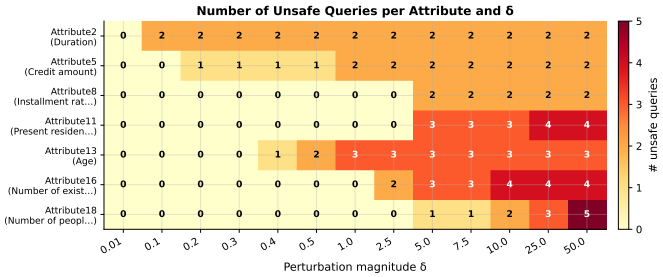


Fig. 12: **Number of unsafe queries per attribute and δ.** The complement of Fig. 11: darker cells indicate more unsafe queries. *Duration* (Attribute 2) accumulates unsafe results earliest and most consistently, confirming it as the attribute to which the model's decisions are most sensitive.

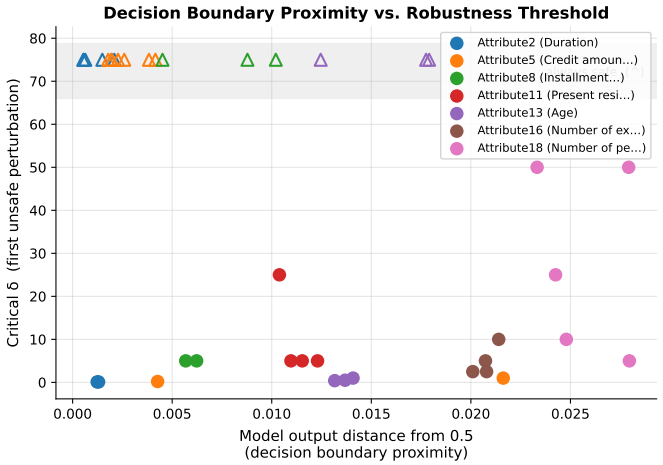


Fig. 13: **Decision-boundary proximity vs. robustness threshold.** Each point is one query; the x-axis is the model's output distance from the 0.5 decision boundary, and the y-axis is the critical δ at which the property first becomes unsafe. Queries for which no unsafe result was found at any tested δ are shown as hollow triangles in the shaded "always safe" band. Points colored by attribute reveal that sensitivity depends on both the attribute type and the sample's proximity to the boundary.

TABLE VI: Per-query inference time (seconds) for GroundingDINO and Grounded SAM on the CUB minimal test set (40 queries, CPU).

Method	Mean (s)	Median (s)	Std (s)
GroundingDINO	8.99	9.38	1.48
Grounded SAM	19.38	19.87	1.76