Politecnico di Torino

Politecnico di Torino

INGEGNERIA INFORMATICA (COMPUTER ENGINEERING) LM-32(DM270)

A.a. 2024/2025

Graduation Session Dicembre 2025

Integrating Neuro-Symbolic Reasoning into 3D Point Cloud Semantic Segmentation

Applied on Cultural Heritage

Relatori:

Lia Morra

Francesca Matrone

Francesco Manigrasso

Candidati:

Fabio Gigante

Abstract

Semantic segmentation of 3D point clouds is a fundamental problem in computer vision, particularly relevant to the digital preservation and automated analysis of cultural heritage. Recent deep learning models such as Point Transformer v3 have achieved state-of-the-art performance by leveraging attention mechanisms to capture both local geometry and long-range spatial relationships. However, purely data-driven approaches still struggle to generalize in geometrically complex or underrepresented classes, and they offer limited interpretability.

This thesis explores a neuro-symbolic framework that integrates logical reasoning into the learning process of a transformer-based point cloud segmentation model. The proposed method combines the representational capacity of deep neural networks with the interpretability and domain-awareness of Logic Tensor Networks (LTN), implemented through the LTNTorch library. Symbolic knowledge about architectural structures—such as coplanarity, verticality, and spatial relationships between classes—is encoded as first-order logic rules and translated into differentiable constraints that guide learning under a best-satisfiability objective. This paired with standard empirical risk minimization provides a regularization effect, encouraging predictions that are both data-consistent and logically coherent.

Experiments are conducted on the ARCH dataset, a large-scale benchmark of annotated architectural point clouds. Point Transformer v3 is trained under standard empirical risk minimization and achieves a mean Intersection-over-Union (mIoU) of 77.5% on the SMV test scene and 71.8% on the SMG scene, outperforming previous DGCNN-based benchmarks by over 20 percentage points. Motivated by these strong results, the study then extends the analysis to the model trained on the complete version of the ARCH dataset, where performance decreased noticeably. A detailed per-class and qualitative analysis revealed semantic inconsistencies in the predictions—for example, violations of spatial and structural coherence between neighboring elements—that persist even in high-performing categories. These inconsistencies motivated the introduction of symbolic priors to enforce domain-consistent reasoning during training.

The neuro-symbolic variant integrates domain rules as trainable predicates, designed to assess whether structured symbolic knowledge can complement data-driven representations and promote logically consistent predictions.

Overall, this work establishes a reproducible framework for evaluating neuro-symbolic reasoning in 3D point cloud segmentation, building upon a thorough assessment of a state-of-the-art model on the ARCH dataset, and outlining promising directions for explainable scene understanding in digital heritage.

Acknowledgements

Voglio ringraziare Defne per essere stata al mio fianco nei momenti belli e brutti di questo percorso, per avermi supportato nei momenti di crisi e per l'infinita pazienza di cui dispone. I miei genitori per avermi supportato durante tutta questa lunga carriera universitaria che mi ha messo alla prova sotto più punti di vista. Mio fratello Emilio per i suoi consigli e punti di vista.

Inoltre sono consapevole che il riconoscimento più grande che si possa desiderare è quello di rendere fiere le persone che ami, e le parole di contentezza di mia nonna Pietrina mi hanno rassicurato sul fatto che ne è valsa la pena.

Table of Contents

List of Tables

List of Figures

Chapter 1

Introduction

The field of Artificial Intelligence (AI) has witnessed remarkable advancements in recent years particularly in the use of sub-symbolic machine learning. While the current era of the field can be described as the third 'AI summer' characterized by rapid advances in research and widespread commercialization, a central development in this phase is the emergence of Neuro-Symbolic AI, which integrates symbolic reasoning with sub-symbolic machine learning [1]. While there remains debate over its necessity —with critics emphasizing the sufficiency of large-scale data-driven approaches and others stressing the need for structured reasoning— this paradigm combines the learning capabilities of deep neural networks with the structured reasoning of symbolic logic, enabling models to generalize from data while adhering to explicit rules and constraints. One promising application of NeuroSymbolic AI is in **Cultural Heritage preservation**, where accurate and interpretable AI models can assist in the analysis, restoration, and documentation of historical artifacts and sites. A critical task in this domain is **3D semantic segmentation**, particularly when dealing with **point cloud data**, which provides a detailed geometric representation of objects and environments. Point cloud semantic segmentation plays a pivotal role in digitalizing cultural heritage, enabling the classification of architectural elements, sculptures, and archaeological findings at a granular level. State-of-the-art deep learning models, such as **Point Transformer V3** [2], have demonstrated exceptional performance in this task by leveraging self-attention mechanisms to capture long-range dependencies in 3D space. However, purely data-driven approaches often lack interpretability and may struggle in scenarios where labeled data is scarce or where domain-specific constraints must be enforced. To address these challenges, this thesis explores the integration of **Logic Tensor Networks (LTNs)** [3] —a NeuroSymbolic framework that combines differentiable learning with first-order logic reasoning— into point cloud semantic segmentation.

1.1 Goal

While Point Transformer V3 serves as a high-performance baseline, the proposed approach trains a 3D semantic segmentation model **from scratch**, incorporating logical rules that encode prior knowledge about structural and semantic relationships in cultural heritage objects through a Neuro-symbolic framework called LTNTorch. By doing so, the goal is for the model not only to learn from data but also to adhere to domain-specific constraints, improving both accuracy and interpretability.

The contributions of this work include:

- A **NeuroSymbolic framework** for point cloud segmentation that integrates deep learning with logical reasoning, extending the functionalities of [4].
- An evaluation of the addition of **Logic Tensor Networks** in a state-of-the-art neural architecture like Point Transformer V3, with a case study on the Arch Dataset [5], demonstrating how logical rules can enhance segmentation performance and interpretability in scenarios with complex and variable geometric structures.

By bridging neural and symbolic AI, this research aims to advance the field of 3D semantic segmentation by analyzing the integration of LTNTorch as a means to build a more robust and explainable framework for cultural heritage preservation.

1.2 Structure of the thesis

The thesis is structured as follows:

1. **Background**: introduces the fundamental concepts of point cloud data, Neuro-Symbolic AI, and Logic Tensor Networks, providing the theoretical foundations necessary to contextualize the proposed approach.
2. **State of the Art**: reviews the current literature in Neuro-Symbolic learning and 3D point cloud semantic segmentation, with particular attention to the ARCH dataset, which constitutes the benchmark for this work.
3. **Methods**: defines the segmentation task, presents the full processing pipeline, and details the construction of the logical theory integrated into LTNTorch.
4. **Experiments**: presents the dataset and analyzes all experimental results, including comparisons with the state-of-the-art models, baseline studies, ablation analyses, and the evaluation of knowledge-based rules.
5. **Conclusions**: summarizes the contributions of the thesis and outlines directions for future work.

Chapter 2

Background

2.1 Point Cloud Data and Semantic Segmentation

A point cloud is an unstructured 3D data representation of the world, typically collected by LiDAR sensors, stereo cameras, or depth sensors. It comprises a collection of individual points, each defined by spatial coordinates in $\mathbb{R}^3$

Semantic segmentation of point clouds is the task of assigning a semantic label to each individual point in a 3D scene; let

$$P = \{p_i \mid p_i \in \mathbb{R}^3,\ i = 1, \ldots, N\}$$

be a point cloud consisting of N points, where each point $p_i = (x_i, y_i, z_i)$ represents its spatial coordinates. Depending on the acquisition device and preprocessing, additional attributes such as color (r_i, g_i, b_i), surface normal vectors $(n_{x_i}, n_{y_i}, n_{z_i})$, or intensity values I_i may be available. The goal is to learn a function that maps each point to a semantic class label chosen from a predefined set of categories:

$$f_\theta : \mathbb{R}^d \to \mathcal{C}$$

where d denotes the dimensionality of the point's feature vector, and

$$\mathcal{C} = \{c_1, c_2, \ldots, c_K\}$$

is the set of K predefined semantic categories.

In the following sections, i outline the foundational principles of symbolic and statistical AI, and introduce neuro-symbolic frameworks—such as Logic Tensor Networks—that aim to unify these paradigms. Such models provide a powerful means of incorporating explicit logical constraints into continuous learning systems, offering a promising direction for structured and interpretable 3D scene understanding.

2.2 Symbolic and Statistical AI

Symbolic AI is a branch of Artificial Intelligence which goal is to learn the internal symbolic representation of the world in scope. It uses symbols, rules and logic to represent knowledge and enable reasoning, problem-solving and decision-making. The knowledge is represented in a declarative fashion, concepts and their relationships represent expert domain knowledge, common-sense knowledge or a semantic web. Reasoning is performed through efficient combinatorial search and it can be used for theorem proving, entailment or planning. In this setting we can translate implicit human knowledge into a more formalized and declarative form, based on rules and logic. In this category fall some of the earliest AI systems, such as the Logic Theorist, the General Problem Solver (GPS) and expert systems like MYCIN. Instead, Statistical AI is a field of AI in which knowledge is simplified to labels, knowledge representation is in most cases sub-symbolic and distributed while the system learns to infer labels through a mathematical mapping function from the input to the output space. This category encompasses approaches such as machine learning, deep learning and generative AI, which rely on algorithms to automatically extract patterns from raw data to discern relationships and make predictions based on learned representations. Examples of some of the earliest AI systems that utilized statistical and sub-symbolic AI include the Perceptron and the ADALINE. The approach evolved over time, achieving impressive results in the most diverse fields, from computer vision to natural language processing. Both approaches have their strength and weaknesses. In summary, symbolic AI excels at incorporating domain knowledge, offering human-readable and highly explainable models, supporting systematic generalization and compositionality when relevant knowledge is available. However, there is an inherent tradeoff between interpretability and scalability: highly expressive symbolic systems enable rich reasoning but can become computationally expensive, and once they grow in size or complexity, the resulting structures may become difficult for humans to interpret. On the other hand, statistical AI can learn directly from examples, is robust to imperfect or partial knowledge, generalizes well within a domain, and is highly scalable due to simple core algorithms. It is particularly effective at representation learning. Nevertheless, statistical AI lacks clear mechanisms for incorporating domain constraints, is sensitive to biases and shortcuts, requires large amounts of data, and cannot easily generalize from few examples. Its systematic generalization is limited, and explainability remains a challenge, often requiring post-hoc methods.

2.3 Neuro-symbolic AI

The idea behind neuro-symbolic AI is to combine the strengths of both symbolic and statistical AI to create more robust, flexible, and interpretable systems. Neuro-symbolic AI aims to leverage these complementary paradigms so that models can learn from data while also reasoning over explicit knowledge representations. It encompasses a family of approaches that can be categorized in different ways. One possible categorization, introduced by Henry Kautz in his essay *The Third AI Summer* [6], is the Kautz Taxonomy, which describes six possible designs for integrating symbolic reasoning and neural networks. These designs vary according to where symbolic and neural components appear, and whether one acts as a subroutine, coroutine, or embedded module within the other.

The taxonomy begins with the $Symbolic \rightarrow Neuro \rightarrow Symbolic$ design, which corresponds to the standard workflow of modern deep learning: symbolic inputs are converted into vectors, processed by a neural network, and then mapped back to symbolic categories or sequences via a softmax layer.

The next class of architectures includes the $Symbolic[Neuro]$ design, in which a neural pattern-recognition module is used as a subroutine within a symbolic problem solver. A notable example is the Go engine *AlphaGo*.

This is followed by the $Neuro \mid Symbolic$ design, where a neural network converts non-symbolic inputs—such as pixels or raw text—into symbolic data structures (e.g., concepts or logical predicates), which are then processed by a symbolic reasoning system. The symbolic component provides feedback that guides the training of the neural network.

The next approach, $Neuro : Symbolic \implies Neuro$, retains the standard neural network architecture but employs a special training scheme based on symbolic rules. Here, symbolic constraints are encoded as training examples, encouraging the network to internalize these rules in its parameters and generalize to unseen inputs.

A further design is the $Neuro_{Symbolic}$ approach, where symbolic rules are transformed into structural templates within the neural network itself.

Finally, the taxonomy includes the $Neuro[Symbolic]$ design, in which a symbolic reasoning engine is embedded inside a larger neural system, mirroring Kahneman's "System 1 / System 2" dual-process theory. In this architecture, the neural network handles fast, intuitive processing, while the symbolic module performs slower, deliberate reasoning tasks.

The approach adopted in this work falls within the $Neuro_{Symbolic}$ family. In particular, we employ Logic Tensor Networks (LTNs) [7], which will be described in the following section.

2.3.1 Abductive reasoning

Before delving into Logic Tensor Networks, it is interesting to understand the concept of reasoning in AI systems; a key difference that we can observe in the symbolic and statistical approach is the way they perform reasoning. Symbolic AI relies on deductive reasoning, which starts from general premises and derives specific conclusions. It is a sound and complete form of reasoning, meaning that if the premises are true, the conclusion must be true as well. Statistical AI, on the other hand, relies on inductive reasoning, which starts from specific observations and derives general conclusions. It is a probabilistic form of reasoning, meaning that the conclusion is not guaranteed to be true, even if the observations are true.

Abductive reasoning instead is a form of reasoning that starts from observations and derives the most likely explanation for those observations, based on existing knowledge and experience. It is a form of reasoning that is often used in scientific inquiry, where scientists observe phenomena and then develop hypotheses to explain those phenomena. In the context of neuro-symbolic AI, and in particular Logic Tensor Networks, reasoning can be seen as a form of abduction, in which the prior knowledge acts as constraints on the learning process, guiding the model to learn from data. This process goal is to restricts the search space of possible solutions, making it compatible with the prior domain knowledge.

2.4 Logic Tensor Networks

In this section Logic Tensor Networks (LTNs) are introduced, along with the concept of Real Logic, the differentiable fuzzy logic on which LTNs are based, to give a complete understanding of the framework.

"Logic Tensor Network is a NeuroSymbolic framework that supports querying, learning and reasoning with both rich data and abstract knowledge about the world" [7]. Its based on a infinitely-valued fuzzy logic, completely differentiable, called Real Logic.

2.4.1 Real Logic

Real Logic is defined on a first-order language $\mathcal{L}$, that contains:

- a set $\mathcal{C}$ of constant symbols (objects),
- a set $\mathcal{F}$ of functional symbols,
- a set $\mathcal{P}$ of relational symbols (predicates),
- a set $\mathcal{X}$ of variable symbols

which form the signature of the language.

Terms are built from constants, variables, and function symbols, and they form the basic components used to construct formulas. Predicates can be applied to an appropriate number of terms to form atomic formulas, which express relational information about variables and constants.

Real Logic is a typed logic, meaning that every symbol in the language is associated with a type that specifies the domain of values it can take. We therefore assume a non-empty set of domains $\mathcal{D}$. Each symbol of the language $\mathcal{L}$ is characterized by an input type domain and an output type domain, both of which are subsets of $\mathcal{D}$.

Semantics in Real Logic To define semantics of $\mathcal{L}$ we need to interpret domains via assignment to tensors in the real field. Since the logic is typed each term is interpreted as a tensor of real values, while functions are interpreted as mappings between tensors and predicates are interpreted as mappings from tensors to truth values in [0,1]. The word used to denote 'interpretation' in Real Logic is 'grounding', while the operation of replacing variables with actual values is called 'instantiation'. The grounding of variables, differently from first-order languages, is done by assigning them to a sequence of values in their domain, instead of a single value. Given this behaviour a term height(x) is also grounded in k different height values, each corresponding to one instance. This means that a term containing n different variables, each one composed of variable instances, namely k_{v_i} instances for variable v_i, is grounded to a tensor of shape

$$k_{v_1} \times k_{v_2} \times \ldots \times k_{v_n} \times d$$

where d is the dimension of the output domain of the term (in the case of a predicate $d = 1$). The dimension associated with d can be called the feature dimension; in the case of predicates it is always 1, since they output truth values. Here is an illustration of grounding taken from [7].

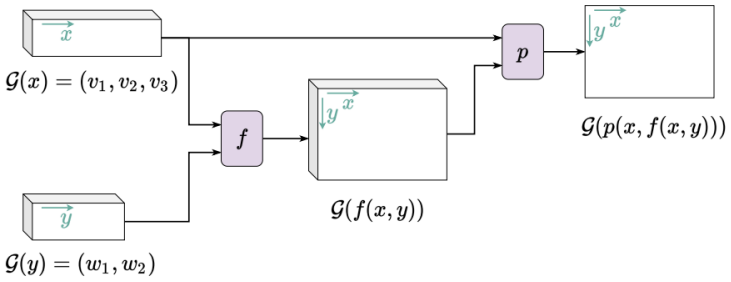


Figure 2.1: Grounding of terms and predicates in Real Logic.

Semantics of Connectives In Real Logic the semantics of logical connectives follow the principles of first-order fuzzy logic. Each connective is associated with a corresponding fuzzy operator:

- conjunction ($\wedge$) is interpreted by a t-norm T,
- disjunction ($\vee$) by a t-conorm S,
- implication ($\Rightarrow$) by a fuzzy implication I,
- negation ($\neg$) by a fuzzy negation N.

These operators, collectively denoted as $\text{FuzzyOp} = \{T, S, I, N\}$, map grounded truth values in [0,1] to new grounded truth values.

Let φ and ψ be formulas with free variables, possibly sharing k of them. For a unary connective α and a binary connective β, grounding is defined as:

$$G(\alpha\varphi) = \text{FuzzyOp}_\alpha\Big(G(\varphi)\Big),$$

$$G(\varphi\ \beta\ \psi) = \text{FuzzyOp}_\beta(G(\varphi),\, G(\psi))\,,$$

where grounding tensors are broadcast and aligned along axes corresponding to shared variables. In grounding a binary connective, the fuzzy operator is applied element-wise to all compatible combinations of instances of the free variables.

Semantics of Quantifiers Quantifiers are defined through aggregation operators. Let Agg be a symmetric, continuous operator:

$$\text{Agg} : [0,1]^n \to [0,1]^{n-h}.$$

For a formula $\varphi(x_1, \ldots, x_n)$ and a quantifier Q applied to the first h variables, grounding proceeds by aggregating the corresponding axes of the grounding tensor. This reduces the dimensionality of the grounding by collapsing the quantified variables.

Diagonal Quantification Logic Tensor Networks also support *diagonal quantification*, written:

$$Q\,\text{Diag}(x_1, \ldots, x_h)\,\varphi.$$

Assuming the variables $x_1, \ldots, x_h$ have groundings with the same number of instances, diagonal quantification aggregates only the diagonal entries of the grounding tensor:

$$G(Q\,\text{Diag}(x_1, \ldots, x_h)\,\varphi) = \text{Agg}_Q(G(\varphi)_{i,\ldots,i})\,.$$

Thus, instead of evaluating φ over all combinations of instances of the variables, the evaluation uses only aligned tuples: the i-th instance of each variable is evaluated together. For example, given samples x and corresponding labels y with matching instance counts, the expression

$$\forall \mathrm{Diag}(x, y)\ p(x, y)$$

checks p only on each pair (x_i, y_i), rather than on all $|X| \times |Y|$ combinations. Diagonal quantification is particularly useful when grounding represents aligned sample–target pairs or synchronized sequences.

Stable Configuration Not all fuzzy operators are equally suitable for gradient-descent. Some choices can lead to numerical instability or poor convergence during optimization. Given a and b, truth values in [0,1], the most effective configuration of fuzzy operators for the purpose is called Product Real Logic and is defined as:

$$\begin{aligned} Not(a) &= 1 - a, & And(a, b) &= ab, \\ Or(a, b) &= a + b - ab, & Implies(a, b) &= 1 - a + ab. \end{aligned}$$

While given n truth-values in [0,1], universal quantification is approximated by:

$$A_{pME}(a_1, \ldots, a_n) = 1 - \left(\frac{1}{n} \sum_{i=1}^{n} (1 - a_i)^p \right)^{1/p}, \qquad p \geq 1,$$

which behaves as a smooth minimum, corresponding to $1 - RMSE(a,1)$ when $p = 2$ and approaching $\min(a_1, \ldots, a_n)$ as $p \to \infty$. Where $RMSE(a,1)$ is the root mean square error between a and 1.

Since this configuration may still suffer from numerical instability during optimization, its operators are modified by projecting truth values slightly away from the boundaries of [0,1]:

$$\pi_0(a) = (1 - \varepsilon)a + \varepsilon, \qquad \pi_1(a) = (1 - \varepsilon)a, \qquad \varepsilon > 0.$$

These projections define the *Stable Product Real Logic* operators:

$$\begin{aligned} Not'(a) &= Not(a), & And'(a, b) &= And(\pi_0(a), \pi_0(b)), \\ Or'(a, b) &= Or(\pi_1(a), \pi_1(b)), & Implies'(a, b) &= Implies(\pi_0(a), \pi_1(b)), \end{aligned}$$

$$A'_{pME}(a_1, \ldots, a_n) = A_{pME}(\pi_1(a_1), \ldots, \pi_1(a_n)).$$

These stabilized operators improve numerical behavior while preserving the structure of Product Real Logic.

2.4.2 Learning Real Logic, Logic Tensor Networks

In the Logic Tensor Networks framework, symbolic rules and neural components are combined, and the learning procedure must ensure that the neural parameters evolve in a way that respects both data and logical constraints. To do so we need a parametric grounding for symbols, denoted as $G(\cdot \mid \theta)$, where θ represents the set of learnable parameters. In this way we have the possibility to ground learnable constants, functions, and predicates. The latter can be given by a neural network $\mathcal{N}$, with parameters θ_N, that maps input tensors to truth values in [0,1]. In the case of semantic segmentation $\mathcal{N}$ takes as input a sequence of point features x_i and outputs a sequence of vectors of class probabilities $\hat{y}_i = (y_{c_1}, y_{c_2}, ..., y_{c_{nc}})$ in $[0,1]^{nc}$, where nc is the number of classes. So $\hat{y} = \mathcal{N}(x \mid \theta_N)$.
Classical neural networks are trained under Empirical Risk Minimization (ERM), where losses such as Cross-Entropy are minimized over examples. Logic Tensor Networks (LTNs), instead, reformulate the objective as a *best-satisfiability* problem: the aim is to adjust the parameters so that the logical theory is satisfied to the highest possible degree.

A Real Logic theory is defined as $T = \langle \mathcal{K}, G(\cdot \mid \theta), \Theta \rangle$, where $\mathcal{K}$ is the set of formulas built from the language symbols, $G(\cdot \mid \theta)$ is the parametric grounding that interprets all symbols and connectives, and Θ is the parameter space. Learning then becomes the problem of finding the parameters θ^* that maximize the overall satisfiability of the theory:

$$\theta^* = \arg\max_{\theta \in \Theta} \; SatAgg_{\phi \in \mathcal{K}} \; G_\theta(\phi). \tag{2.1}$$

The operator $SatAgg$ aggregates the truth values of all formulas into a single score expressing the consistency of the grounded theory. It is defined in $[0,1]* \to [0,1]$ and is implemented into the framework as a generalized mean with respect to error, A_{pME}:

$$SatAgg_{\phi \in \mathcal{K}} \; G_\theta(\phi) = 1 - \left(\frac{1}{n} \sum_{i=1}^{n} (1 - P(\phi_j))^p \right)^{\frac{1}{p}} \tag{2.2}$$

where $P(\phi_j) \in [0,1]$ denotes the truth degree of formula ϕ_j given the grounding, and p is a parameter controlling the aggregation behavior (e.g.,$p = 1$ for average, $p \to \infty$ for minimum), typically set to 2.

Chapter 3

State of the Art

3.1 Point Cloud Semantic Segmentation

Semantic segmentation of 3D point clouds has evolved from early point-based neural networks toward graph-based and Transformer-based architectures. According to the recent survey [8], Transformer-based models currently represent one of the most impactful directions.

Transformers employ *self-attention*, that dynamically learns interactions between points. Several architectures exploit this idea: channel self-attention models [9], hybrid attention–GCN methods such as AGCN [10] and variants incorporating enhanced positional encodings like Point Transformer v2 [11].

A key challenge highlighted by the survey is the computational cost of self-attention, which scales quadratically with the number of points. To address this, several efficient alternatives have been proposed, including low-rank approximations, lightweight localized attention, and window-based models such as Stratified Transformer [12]. These approaches restrict or approximate attention while preserving contextual reasoning.

Within this landscape, the *Point Transformer* family has emerged as a leading architecture. The original Point Transformer [13] introduced MLP-based positional encoding within vector attention, while Point Transformer v2 [11] strengthened geometric encoding and introduced partition-based pooling. **Point Transformer v3** [2], the most recent variant, directly addresses efficiency concerns by replacing expensive KNN searches with an *efficient serialized neighborhood mapping* to group points, removing the need of neighborhood mechanism in the attention module. In addition it replaces relative positional encoding with a simpler sparse convolutional layer. This changes allow the expansion of the attention patch size from 16 to 1024 points while remaining computationally tractable, achieving a favorable balance between accuracy, simplicity, and scalability. This makes Point Transformer v3

well suited for large cultural heritage scenes, which are dense, irregular, and geometrically complex.

3.2 Neuro Symbolic Learning

The neuro-symbolic learning approach adopted in this thesis corresponds to what the survey [14] defines as *"reasoning for learning."* In this paradigm, neural networks perform a machine learning task while symbolic knowledge is incorporated directly into the training process to improve both performance and interpretability.

Within this category, two main architectural families can be identified:

1. **Regularization-based models**, in which symbolic knowledge constrains the learning process through an additional regularization term in the loss function. As discussed in Section 2.4, this term is *grounded* from a knowledge base that encodes the relevant symbolic information.

2. **Knowledge-transfer models**, which leverage symbolic knowledge to guide learning in a target domain that differs from the domain used for training.

In the present work, we focus on the former category, where symbolic constraints directly shape the optimization dynamics of the neural model.

Examples of Regularization-based frameworks are Semantic-based Regularization (SBR) by Diligenti et al. [15], and the conceptually aligned Logic Tensor Networks (LTN) [7], adopted in this thesis work. Both frameworks integrate symbolic knowledge into the learning process by introducing a differentiable penalty term in the loss function, which enforces the satisfaction of logical constraints. They similarly use a fuzzy translation of first-order logic into real-valued semantics to guide neural learning. In this sense, SBR and LTN share the same foundational principle of using logic-driven regularization to shape the behavior and interpretability of neural models. In particular, the LTN framework makes it possible to express and efficiently carry out a wide range of core AI tasks within a single, unified formalism. These tasks include multi-label classification, relational reasoning, clustering, semi-supervised learning, regression, embedding, and query answering.

3.2.1 Applications

Applications of neuro-symbolic learning are diverse, and several closely relate to the task addressed in this thesis. Early work includes semantic segmentation in remote sensing, dating back to 2013 [16], while more recent developments involve point-cloud semantic segmentation for cultural heritage in 2023 [17]. Although both

approaches incorporate expert knowledge, they differ from the regularization-based methodology adopted in this thesis. The method in [16] begins with the prediction probabilities produced by a baseline segmentor and iteratively refines the regions that show the lowest agreement with expert knowledge. At each step, alternative labels are evaluated using probability-based criteria, and the process continues until predictions converge.

In contrast, KENN operates directly on the neural network's output predictions and applies a sequence of specialized layers to refine them, increasing satisfaction of the encoded knowledge.

Both approaches fundamentally differ from the regularization-based strategy used in this work, where symbolic knowledge directly modify the neural network's weights during training.

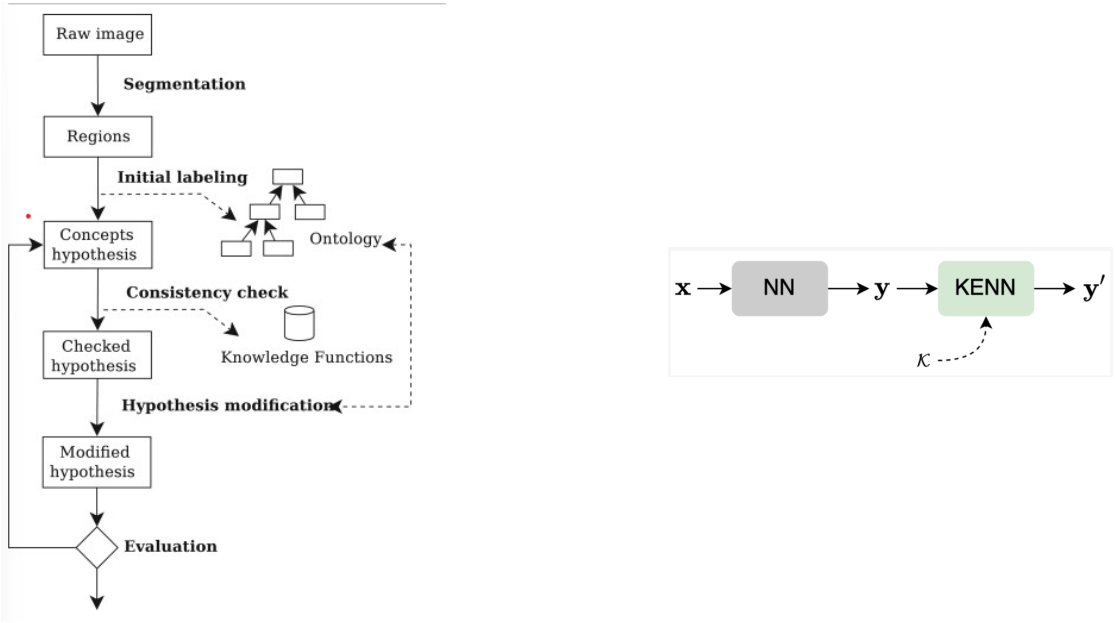


Figure 3.1: Architectures of [16] (left) and KENN (right).

Regularization-based methods Applications for regularization-based neuro symbolic methods include Object and Part detection [18], classification tasks [19] and semantic segmentation [20], where it shows interesting results, adding robustness to the learning framework and improving the results, especially when data is scarce or errors are present in the labels.

3.3 Deep Learning for Cultural Heritage 3D Segmentation

The survey [8] highlights the importance of deep 3D semantic segmentation for *cultural heritage preservation*, where automated analysis of architectural scenes enables accurate documentation, monitoring, and digital reconstruction of historic environments. Within this context, a central reference for this thesis is the work of Matrone et al. [21], which represents one of the first comprehensive evaluations of machine learning and deep learning strategies for large-scale cultural heritage point clouds.

Their proposed architecture, **DGCNN-Mod+3Dfeat**, extends the original DGCNN [22] by integrating handcrafted geometric descriptors—such as normals, planarity, verticality, omnivariance, and other covariance-based features—to improve the segmentation of fine architectural elements. Their study demonstrates that enriching point-wise representations with local geometric structure can significantly benefit classes with thin, irregular geometry.

The present thesis builds directly upon this line of work by replacing the handcrafted-feature–dependent DGCNN with a modern Transformer-based architecture (Point Transformer v3), capable of learning local and global geometric relations end-to-end. This allows an updated and more expressive baseline to be established on the ARCH dataset, ultimately enabling the integration of neuro-symbolic 'reasoning for learning' on a stronger geometric backbone.

Chapter 4

Methods

This chapter describes the methodological framework adopted to integrate neuro-symbolic learning into the task of 3D point cloud semantic segmentation. The proposed approach combines the representational power of deep neural networks with the interpretability and domain-awareness of logical reasoning, leveraging the LTNTorch library as a differentiable reasoning engine.

The goal of this methodology is, in a first instance, to train a state of the art model in the task of point cloud semantic segmentation within a complex dataset in the field of cultural heritage, to assess the capability of sub-symbolic models of understanding and inferring semantic labels in 3D point clouds. In a second instance to train the same model under logical constraint to assess the capability of neuro-symbolic models to improve the segmentation performance by leveraging symbolic knowledge in this task.

The remainder of this chapter is organized as follows: Section 4.1 formalizes the segmentation problem in a neuro-symbolic context. Section 4.2 presents the data processing pipeline, composed of the baseline network configuration and the neuro-symbolic formulation implemented with LTNTorch.

4.1 Problem definition

To reach our goal of embedding knowledge in the baseline model, through the definition of symbolic rules, we need to change the sub-symbolic objective. For instance we need to move from a learning objective based on Empirical Risk Minimization to an objective based on best satisfiability. As described in 2.4.2 , in Logic Tensor Networks the learning objective is to find the parameters θ^* that maximize the overall satisfiability of the theory:

$$\theta^* = \arg\max_{\theta \in \Theta} \; SatAgg_{\phi \in \mathcal{K}} \, \mathcal{G}_\theta(\phi). \tag{4.1}$$

When training is performed on mini-batches, this objective is adapted to stochastic optimization. For a batch $\mathcal{B}$ of grounded samples and the set of clauses $\mathcal{K}$, the loss function is written as:

$$\mathcal{L} = 1 - SatAgg_{\phi \in \mathcal{K}} \, \mathcal{G}_{\theta, \, x \in \mathcal{B}}(\phi), \tag{4.2}$$

so that minimizing $\mathcal{L}$ corresponds to maximizing the satisfiability of the theory. In this way, standard ERM losses can be incorporated.

Loss Function

In an effort to maintain the objective similar to the classic supervised learning setting, to enable comparison, we can use an hybrid loss function that combines domain specific knowledge with supervised loss:

$$\mathcal{L} = \mathcal{L}_{CE} + \mathcal{L}_{Lovasz} + \lambda \mathcal{L}_{domain} \tag{4.3}$$

where $\mathcal{L}_{CE}$ and $\mathcal{L}_{Lovasz}$ are respectively the cross entropy and the Lovasz loss functions, both defined in the baseline, while $\mathcal{L}_{domain}$ is the objective defined before and λ is a hyperparameter that controls the weight of the domain knowledge loss.

4.2 Pipeline Overview

This section presents the complete processing and learning pipeline developed for applying neuro-symbolic learning to point cloud semantic segmentation. The proposed pipeline integrates both geometric and logical components combining a deep learning backbone for feature extraction and semantic prediction with domain-specific constraints through differentiable logic.

The overall workflow consists of three main stages. In the first stage, point clouds are preprocessed to make them suitable for forwarding. In the second stage, a semantic segmentation model based on Point Transformer v3 is trained under one of two alternative configurations:

- a standard data-driven setting, where the network is optimized using only conventional loss functions, namely CrossEntropy and Lovasz;
- a neuro-symbolic setting, where the model is trained incorporating logical predicates and rules defined in LTNTorch to impose soft constraints during learning.

The resulting predictions are reconstructed at full resolution and quantitatively evaluated using standard segmentation metrics.

The following subsections describe each component of this pipeline in detail, including preprocessing and feature extraction, baseline model configuration, integration of LTNTorch predicates, and evaluation of the final segmentation outputs.

4.2.1 Preprocessing & Feature Extraction

In our case, the point cloud structure is as follows: each point is represented by its spatial coordinates (x, y, z), color attributes (r, g, b), a semantic label, and surface normal components (N_x, N_y, N_z). These features were extracted and preprocessed as described in the existing work on the dataset [5] to ensure consistency and suitability for neural network processing.

In addition, in some experiments, the same 3D features used in a related study on the dataset [21] were also incorporated. These features, inspired by prior works [23, 24], are derived from the eigenvalues of the covariance matrix computed over local neighborhoods in the point cloud. Specifically, verticality, omnivariance, surface variation, planarity, and a normalized z scalar field were extracted to capture fine geometric properties of architectural elements. In some cases, the radii were fine-tuned for the specific scene in order to better adapt to local geometric density and scale variations. and stacked in the input vector, along with rgb and normals. These 3D features were used to evaluate their impact on segmentation performance, comparing baseline models with versions augmented by these descriptors.

The overall framework and preprocessing pipeline are adapted from Pointcept [4], ensuring consistency with established best practices in point cloud semantic segmentation.

The preprocessing pipeline for training the semantic segmentation model is designed to augment and normalize the input point clouds, improving generalization and model performance. The pipeline consists of a sequence of geometric and color transformations, applied probabilistically to each sample during training. The most important one is GridSample, in which the point cloud is voxelized, enforcing spatial regularity and allowing serialization.

During validation and testing all the probabilistic augmentations are disabled and only normalization and grid sampling are applied to ensure consistent evaluation.

4.2.2 Baseline

The adopted semantic segmentation model is **Point Transformer v3 (PTV3)** [2], which represents the state-of-the-art in point cloud segmentation due to its ability to model both local geometric relationships and long-range dependencies. It employs a transformer-based encoder-decoder architecture specifically designed for unstructured 3D data.

Architectural Overview.

PTV3 introduces a hierarchical framework that alternates between grid pooling and transformer-like blocks to progressively aggregate spatial context. The network begins with a voxelization and **serialization** stage, where points are encoded and

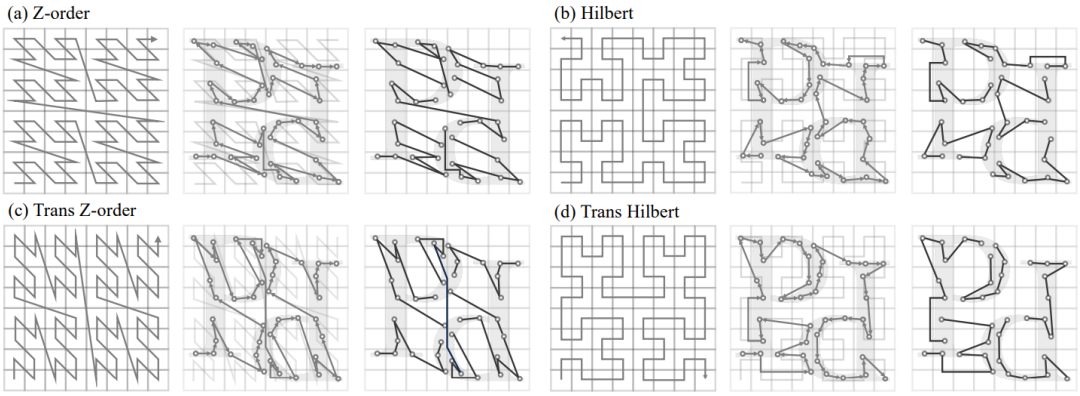


Figure 4.1: Overview of serialization and patch grouping, which includes creation of serialization curve (left), ordering in respect to the serialization curve (middle), patch grouping to prepare for local attention (right)

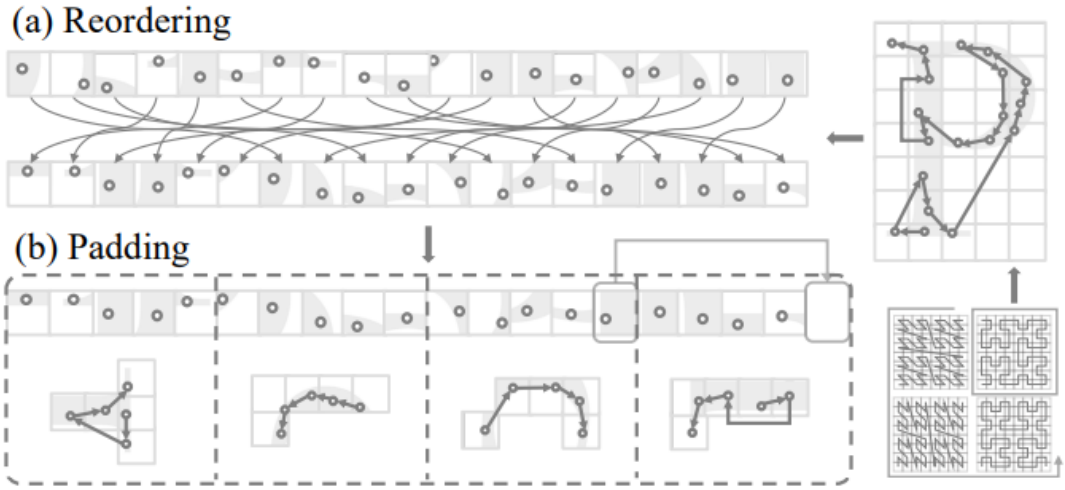


Figure 4.2: Overview of Point Transformer v3 (PTV3) serialization stage, which includes, in order,the selection of the serialization order (that is changed sequentially on each attention layer of the encoder stage), point cloud serialization and ordering, and finally padding of the last sequence by borrowing points from neighboring patches,to ensure it is divisible by the designated patch size.

ordered in multiple ways using various space-filling curves (Z-order, Z-transformed, Hilbert,Hilbert-transformed) as shown in figure 4.1. This serialization defines a spatial ordering that allows the model to partition the point cloud into fixed-size patches. Each point retains both its serialized index and inverse mapping, enabling bidirectional transitions between original and serialized coordinates. A scheme of

the process is shown in figure 4.2

Following serialization, the **encoder** is composed of five hierarchical stages, each consisting of a `SerializedPooling` layer and several transformer `Block` modules. Each block integrates a conditional positional encoding (based on sparse 3D convolutions), a multi-head **SerializedAttention** mechanism, and a feed-forward MLP. Serialized attention computes self-attention within localized point patches, functioning similarly to windowed attention in vision transformers, but in this case the patch boundaries depend on the serialization step. Successive pooling stages increase the receptive field by merging neighboring points in serialized space, allowing high-level stages to encode broader spatial structures while maintaining computational efficiency.

The **decoder** mirrors the encoder hierarchy through `SerializedUnpooling` layers, which restore the spatial resolution of points by inverting the pooling operations using the saved parent-child mappings. Each decoder stage fuses upsampled features with corresponding skip connections from the encoder, followed by additional transformer blocks that refine semantic information. The final segmentation head is a simple linear classifier that maps the decoder output features to the target number of classes.

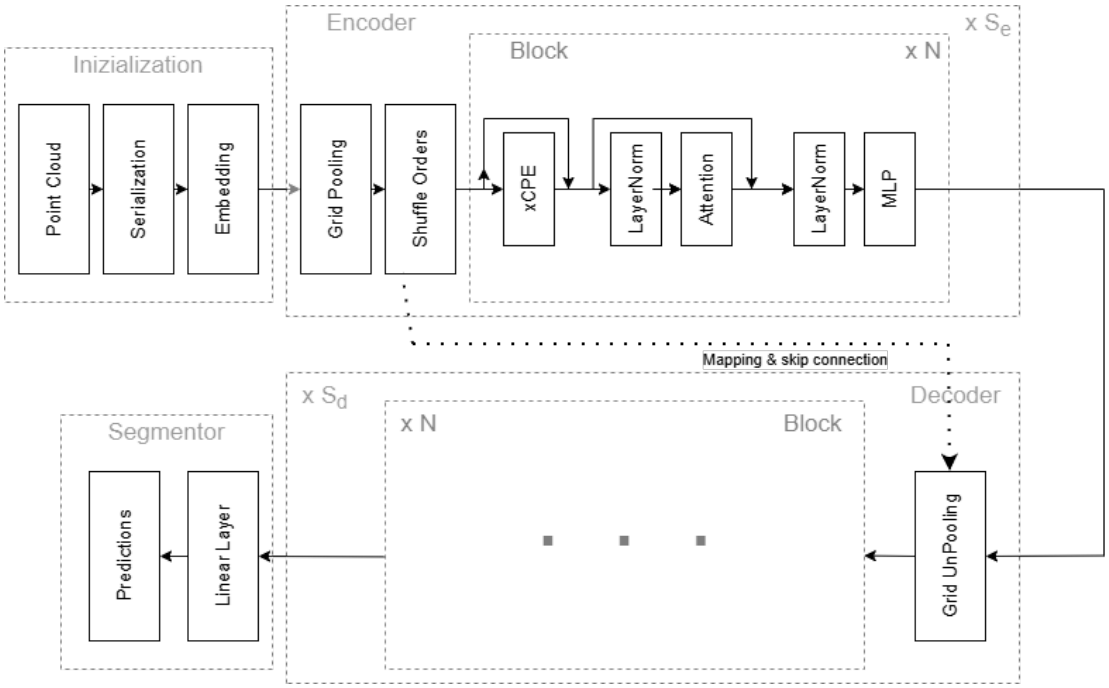


Figure 4.3: Overview of Point Transformer v3 (PTV3) overall architecture

A diagram of the overall architecture is shown in figure 4.3.

Loss Function

For training the baseline model on the point cloud semantic segmentation task, we follow the loss formulation adopted by the original authors. The overall objective consists of two components: the standard Cross-Entropy Loss and the Lovász Loss. Both terms contribute equally to the optimization, and the total loss is computed as a unit-weighted sum of the two.

Cross-Entropy Loss Cross-Entropy Loss provides point-wise supervision and is widely used for multi-class classification. For a point with predicted class probability distribution $\mathbf{p}$ and ground-truth label y, the loss is defined as

$$\mathcal{L}_{\text{CE}} = -\log p_y.$$

Given a point cloud with N points, the aggregated loss becomes

$$\mathcal{L}_{\text{CE}} = -\frac{1}{N}\sum_{i=1}^{N} \log p_{i,y_i},$$

where y_i is the ground truth label of point i. Encouraging the network to assign high probability to the correct semantic class for each point.

Lovász Loss The Lovász Loss is a differentiable surrogate of the Intersection-over-Union (IoU) [25], making it well-suited for segmentation tasks in which region-level accuracy is more meaningful than point-wise accuracy alone. The Lovász-Softmax formulation computes the Lovász extension of the IoU over the sorted point-wise prediction errors. For a class c with error vector $\mathbf{m}^{(c)}$ sorted in decreasing order, the loss can be expressed as

$$\mathcal{L}_{\text{Lovasz}} = \frac{1}{C}\sum_{c=1}^{C} \overline{\Delta_{\text{IoU}}}\left(\mathbf{m}^{(c)}\right),$$

where $\overline{\Delta_{\text{IoU}}}$ denotes the Lovász extension of the IoU. This loss directly promotes improvements in IoU by penalizing the points that most strongly affect class-level segmentation performance.

Total Loss Since both terms are used with unitary weight, the final loss is simply

$$\mathcal{L} = \mathcal{L}_{\text{CE}} + \mathcal{L}_{\text{Lovasz}}.$$

This combination offers a balance between stable point-wise supervision and IoU-oriented region-level optimization, which is crucial for achieving high-quality point cloud semantic segmentation.

Model Configuration.

Throughout this work, the model was configured following the original configuration [2], with encoder depths of (2,2,2,6,2), encoder channels (32,64,128,256,512), and head dimensions (2,4,8,16,32). The decoder mirrors this structure with four stages of channels (64,64,128,256) and identical patch sizes of 1024. Each input point is described by six or more features, corresponding to color (r, g, b) and surface normals (N_x, N_y, N_z), optionally extended with handcrafted 3D descriptors such as verticality or omnivariance (see Section 4.2). For optimization, the network employs a combination of CrossEntropy and Lovasz losses with unit weight, optimized with the AdamW optimizer and a OneCycleLR scheduler.

Inference Characteristics and Comparison with DGCNN.

A key methodological distinction between this model and previous benchmarks lies in the inference strategy. In [21], DGCNN was applied to *subsampled point clouds*, where each scene was partitioned into smaller cuboid volumes forwarded independently through the network. Predictions were computed only for the sampled points, and the final outputs were not re-projected to the full-resolution point cloud, resulting in metrics calculated over a reduced subset of points.

In contrast, Point Transformer v3 processes large spatial contexts, without the need of manual partitioning of the scene, thanks to its hierarchical patch-wise attention mechanism. Rather than relying on fixed cuboid, PTV3 attends to fixed-size patches, computed during the serialization step for each scene. Its serialized pooling progressively expands the receptive field, allowing the model to aggregate information from increasingly larger neighborhoods. In our case up to an estimated volume of approximately $33.5\,\mathrm{m}^3$ per patch in the deepest encoder stage [1]. During evaluation, predictions are upsampled to all original points using nearest-neighbor mapping, ensuring that every point in the scene contributes to the computed metrics. This results in smoother and more coherent segmentations, particularly in complex or high-detail regions.

This configuration makes Point Transformer v3 a robust baseline for evaluating the integration of logical constraints through LTNTorch, serving as the foundation for both the standard and neuro-symbolic training regimes described in the following sections.

[1] Using grid size of 0.02, not considering Shift Order mechanism, a mechanism that allows all the different serialization order to be attended sequentially, explained in detail in [2]

4.2.3 Logic Tensor Network

This section will describe the integration of Logic Tensor Networks (LTN) into the semantic segmentation pipeline,focusing on the definition of logical predicates and grounding mechanisms used to incorporate domain knowledge.

A set of logical predicates was defined, derived from domain knowledge. The logical predicates focus on a subset of classes, selected due to their lower segmentation performance compared to other architectural elements, indicating potential areas where domain knowledge could provide significant improvements. The notation that will be used throughout this section can be found in the Background chapter 2.4.

Grounding

Signature and Assignment To define the Real Logic Theory for our problem we need to define the signature and the assignment of the symbols.

Domains

1. **Point** , grounded in $\mathbb{R}^{3+F}$, where F is the number of features per point (e.g., color, normals, handcrafted 3D features). It represents a single point in the point cloud.
2. **Logits**, grounded in $\mathbb{R}^{C}$, where C is the number of classes. They represent the raw output scores from the segmentation model before applying softmax.

Constants

1. *arch*, *moldings*, *column*, *floor*, *door_window*, *wall*, *stairs*, *vault*, *roof*, *other* : grounded in $\mathbb{N}^{10}$, representing the one-hot encoding of each class.

Variables

1. x : point belonging to the point cloud, grounded in $\mathbb{R}^{3+F}$

Predicates

1. verticality(x) represent the degree of verticality of point x, derived from the correspondent descriptor in input and normalized between 0 and 1 using min-max normalization, based on the min and max values in the training set. The same process is applied to the other geometric descriptors.
2. omnivariance(x) represent the degree of omnivariance of point x, derived from the correspondent descriptor in input.

3. planarity(x) represent the degree of planarity of point x, derived from the correspondent descriptor in input.

4. surfaceVariation(x) represent the degree of surface variation of point x, derived from the correspondent descriptor in input.

5. Coplanarity(x,y) represent the degree of coplanarity between points x and y, its defined starting from normals and coordinates of points. The degree of coplanarity is computed as a product of surface coplanarity of point y in respect to x and then multiplied by the normal similarity of the two points. Surface coplanarity is defined as

$$1 - \frac{\|v_{xy} \cdot n_x\|^2}{\|v_{xy}\|^2}$$

where v_{xy} is the vector from point x to point y, and n_x is the normal vector at point x. Normal similarity is defined as

$$\left(\frac{1 + n_x \cdot n_y}{2}\right)^{\alpha}$$

where n_x and n_y are the normal vectors of points x and y respectively, and α is a scaling factor to control the sharpness of the similarity measure. Both metrics are bounded between 0 and 1 since n_x and n_y are normalized.

6. Near(x,y) represent the degree of proximity between points x and y, defined as the gaussian of the euclidean distance between the two points, multiplied by a scaling factor κ to control the spread.

$$\exp(-\kappa * ||x - y||^2)$$

7. GreatherThan(Predicate(x), Predicate(y)) represent the degree to which Predicate(x) is greater than Predicate(y), is defined on the sigmoid of the difference between the two predicates, multiplied by a scaling factor κ to control the steepness.

$$\sigma(\kappa * (p_1(x) - p_2(y)))$$

8. SimilarRGB(x,y) represent the degree of similarity in color between points x and y, defined as the gaussian kernel of the euclidean distance between the rgb values of the two points, multiplied by a scaling factor κ to control the spread.

$$\exp(-\kappa * ||rgb(x) - rgb(y)||^2)$$

9. Arch(x), Moldings(x), Column(x), Floor(x), Door_window(x), Wall(x), Stairs(x), Vault(x), Roof(x), Other(x) : represent the probability of point x belonging to each class, it is obtained by applying sequentially a softmax and gather function over the output logits of the baseline model.

Logical Rules

The rules were defined based on domain knowledge and an analysis of the errors made by the baseline model, focusing on the classes with lower segmentation performance. Each rule will be presented in its logical form, justified by the brief high level concept behind it. A concept to keep in mind while reading the rules is that the predicates describing the grade of belonging to a certain class is directly tied to the current model world representation, since it correspond to the probability given to the class by the baseline.
Rules can be divided into *type 1* and *type 2*. Type 1 rules operate on the features of a single point, while Type 2 rules operate on relationships between neighboring points. There could also be *type 3* rules that operate on characteristics involving more than two points—for example, rules applied to instances of architectural elements—but these were not implemented in this work. Although they could be interesting for future developments. Rules were grouped into sets to achieve a certain general semantic, each group is associated with an abbreviation to facilitate references. Some of the sets were created from general concepts, while others were formulated in response to baseline model limitations or developed incrementally to support reasoning during learning and prevent shortcut solutions from the model.

Type 1 Rules The explanation of this kind of rules is simple, they express low level characteristics of the classes, for example that doors and windows are not characterized by surface variation, or that walls are vertical structures.

- Door Window Rule, **DW**, defines that each point belonging to the door_window class have a low surface variation:

$$\forall x : \texttt{Door\_Window}(x) \Rightarrow \neg\texttt{SurfaceVariation}(x)$$

- Column Rule, **CR**, defines that each point belonging to the column class have a low planarity value:

$$\forall x : \texttt{Column}(x) \Rightarrow \neg\texttt{Planarity}(x)$$

- Extended Column Rule, **ECR**, Since it was observed during the experiments that the model frequently negated the antecedent when rules of the form

$$Class(x) \implies Characteristic(x)$$

were used, a complementary set of rules was created to supplement the existing one. In particular, the rule construction was inverted to

$$Characteristic(x) \implies Class(x)$$

The resulting rule for the *CR* set was:

$$\forall x:\ \neg\texttt{Planarity}(x)\implies\Big(\texttt{Column}(x)\vee\texttt{Arch}(x)\vee\texttt{Door\_Window}(x)\vee\texttt{Other}(x)\Big)$$

derived from domain knowledge and a per class feature analysis of the dataset.

- Arch Rule, **AR**, tries to simplify a concept observed in the data for which arches have high surface variation on their borders, while planarity is in a characteristic medium range and verticality is high on the curvature maximum, while lowering when approaching the curvature minimum. This concept was hard to formalize with the current language, but an attempt was:

$$\forall x:\ Arch(x)\Rightarrow\Big(\texttt{SurfaceVariation}(x)\wedge\neg\Big(\texttt{Planarity}(x)\vee\texttt{Verticality}(x)\Big)\Big)$$

- Molding Rule, **MR**, defines that molding points have an high omnivariance value:

$$\forall x:\ Moldings(x)\Rightarrow Omnivariance(x)$$

- Extended Molding Rule, **EMR**,as discussed in the description of the rule *ECR*, in an effort to prevent shortcut behaviour this set of rules was created:

$$\forall x:\neg\texttt{Omnivariance}(x)\implies\Big(\texttt{Arch}(x)\vee\texttt{Floor}(x)\vee\texttt{Wall}(x)\vee\texttt{vault}(x)\vee\texttt{Roof}(x\Big)$$

- Floor Wall Verticality, **FWV**, formalize the concept that points belonging to Floor have low verticality and points belonging to Wall have high verticality:

$$\forall x:\ \texttt{Floor}(x)\Rightarrow\neg\texttt{Verticality}(x)$$

$$\forall x:\ \texttt{Wall}(x)\Rightarrow\texttt{Verticality}(x)$$

- Extended Floor Wall Verticality, **EFWV**, as discussed in the description of the rule *ECR*, in an effort to prevent shortcut behaviour this set of rules was created:

$$\forall x:\neg\texttt{Verticality}(x)\implies\Big(\texttt{Floor}(x)\vee\texttt{Vault}(x)\vee\texttt{Arch}(x)\vee\texttt{Other}(x)\vee\texttt{Roof}(x)\vee\texttt{Stairs}(x)\Big)$$

$$\forall x:\texttt{Verticality}(x)\implies\Big(\texttt{Wall}(x)\vee\texttt{Column}(x)\vee\texttt{Arch}(x)\vee\texttt{Moldings}(x)\vee\texttt{Door\_window}(x)\vee\texttt{Other}(x)\Big)$$

derived from domain knowledge and a per class feature analysis of the dataset.

Type 2 Rules

- Door Window Pair, **DWP**, this pair of rules aim to express that doors have higher omnivariance than neighboring wall points and that windows have homogeneous color in a local neighborhood, that is different from neighboring walls or moldings. The concept is to enforce this rule in a radius to guide the model into satisfying this local constraint:

$$\forall x, y \in \text{radius} : \Big(\texttt{Door\_Window}(x) \wedge \texttt{Wall}(y)\Big) \Rightarrow \texttt{GreaterThan}\Big(\texttt{Omnivariance}(x),\ \texttt{Omnivariance}(y)\Big)$$

$$\forall x, y \in \text{radius} : \Big(\texttt{Door\_Window}(x) \wedge \texttt{Door\_Window}(y)\Big) \Rightarrow \texttt{Similar\_RGB}(x, y)$$

- Roof Floor Consistency, **RFC**, This group of rules aim to enforce continuity in the locality of planar structures, particularly in floors and roofs, trying to prevent misclassifications with class other and in the case of roofs with class floor.

$$\forall x, y \in \text{radius} : \Big(\texttt{Roof}(x) \wedge \texttt{Coplanarity}(x, y)\Big) \Rightarrow \neg\texttt{Other}(y)$$

$$\forall x, y \in \text{radius} : \Big(\texttt{Floor}(x) \wedge \texttt{Coplanarity}(x, y)\Big) \Rightarrow \neg\texttt{Other}(y)$$

$$\forall x, y \in \text{radius} : \Big(\texttt{Roof}(x) \wedge \texttt{Coplanarity}(x, y)\Big) \Rightarrow \neg\texttt{Floor}(y)$$

- Three variants were formed to complement the former *RFC* rule, called *ADD1*, *ADD2* and *ADD3*
  - **ADD1** States that for all the pairs x,y of points in a radius, if point x is a roof/floor then the neighboring point y is a roof/floor and is coplanar to y. The rule is wrong because it doesn't take into account elements attached to roofs or floors, for examples chimneys, but is kept since it was tested:

$$\forall x, y \in \text{radius} : \texttt{Roof}(x) \Rightarrow \Big(\texttt{Roof}(y) \wedge \texttt{Coplanarity}(x, y)\Big)$$

$$\forall x, y : \texttt{Floor}(x) \Rightarrow \Big(\texttt{Floor}(y) \wedge \texttt{Coplanarity}(x, y)\Big)$$

  - **ADD2** States that for all the pairs x,y of points in a radius, if point x is a roof/floor, all points y that are coplanar to x are roof/floor.

$$\forall x, y \in \text{radius} : \texttt{Roof}(x) \wedge \texttt{Coplanarity}(x, y) \Rightarrow \texttt{Roof}(y)$$

$$\forall x, y : \texttt{Floor}(x) \wedge \texttt{Coplanarity}(x, y) \Rightarrow \texttt{Floor}(y)$$

– **ADD3** States that for all the pairs x,y of points in a radius, if both points are roof/floors then they are coplanar:

$$\forall x, y \in \text{radius}: \ \texttt{Roof}(x) \land \texttt{Roof}(y) \Rightarrow \texttt{Coplanarity}(x, y)$$

$$\forall x, y: \ \texttt{Floor}(x) \land \texttt{Floor}(y) \Rightarrow \texttt{Coplanarity}(x, y)$$

- Extended Roof Floor Consistency, **RFC**, as discussed in the description of the rule *ECR*, in an effort to prevent shortcut behaviour this set of rules was created. The first rule aims to reduce the probability of predicting class *Other* when handling flat and planar structures:

$$\begin{aligned}\forall x, y \in \text{radius}: \texttt{Coplanarity}(x, y) \land \neg\texttt{Verticality}(x) \land \neg\texttt{Verticality}(y) \\ \Rightarrow \neg(\texttt{Other}(x) \lor \texttt{Other}(y))\end{aligned}$$

  The second rule instead is more restrictive and says that two points that lay in the same flat and planar structure must be both Roof or both Floor.

$$\begin{aligned}\forall x, y \in \text{radius}: \texttt{Coplanarity}(x, y) \land \neg\texttt{Verticality}(x) \land \neg\texttt{Verticality}(y) \\ \Rightarrow \Big(\texttt{Roof}(x) \land \texttt{Roof}(y)\Big) \lor \Big(\texttt{Floor}(x) \land \texttt{Floor}(y)\Big)\end{aligned}$$

  The third rule aim to disincentive the model in the prediction of Floor for each point whom z coordinate is over a certain threshold.

$$\forall x: \ \texttt{Elevated}(x, \mathit{threshold}) \Rightarrow \neg\texttt{Floor}(x)$$

4.2.4 Segmentation output and metrics calculation

After obtaining the output logits from the model, the predictions are computed by taking the arg max over the class dimension for each sample.

As explained in the Baseline section (Section 4.2.2), during evaluation the predictions are re-projected and then compared to the ground-truth segment labels to calculate per-class intersection, union, and target counts. The *intersection* represents the number of correctly predicted points for each class, the *union* is the total number of points either predicted or labeled as a given class, and the *target* is the total number of ground-truth points per class.

This three values are accumulated over the entire split and used to compute various segmentation metrics. A small constant ϵ is added to denominators to prevent division by zero.

After all samples of the split have been processed, the following metrics are computed:

- **Per-class IoU:**

$$\text{IoU}_c = \frac{\text{intersection}_c}{\text{union}_c + \epsilon}$$

- **Per-class accuracy:**
$$\text{Acc}_c = \frac{\text{intersection}_c}{\text{target}_c + \epsilon}$$
- **Mean IoU (mIoU):** average of per-class IoUs,
$$\text{mIoU} = \frac{1}{C} \sum_{c=1}^{C} \text{IoU}_c$$
- **Mean accuracy (mAcc):** average of per-class accuracies,
$$\text{mAcc} = \frac{1}{C} \sum_{c=1}^{C} \text{Acc}_c$$
- **Overall accuracy (allAcc):**
$$\text{allAcc} = \frac{\sum_c \text{intersection}_c}{\sum_c \text{target}_c + \epsilon}$$

Additional metrics such as precision, recall, and F1-score were also computed, particularly for the comparison of state-of-the-art. These were obtained using the standard `scikit-learn` library.

4.2.5 Configuration and Reproducibility

All experimental settings, including seed, hyperparameters, data augmentations, optimizer configurations, and training schedules, are specified through modular configuration files following the structure adopted by the Pointcept framework [4]. These files define every component of the pipeline—model architecture, dataset splits, data transformations, optimizer, and learning rate scheduler—ensuring that experiments are fully reproducible and easily modifiable. The modular design also enables switching between training, validation, and test modes through parameterized dataset definitions.This explicit configuration approach allows exact reproduction of all experiments presented in Chapter 5 and facilitates future extensions or ablation studies.

Chapter 5

Experiments

This chapter presents the experimental setup and results obtained by applying the proposed neuro-symbolic learning framework to the task of semantic segmentation of 3D point clouds from the ARCH dataset. The experiments are designed to evaluate the effectiveness of incorporating logical constraints, through LTNTorch [26], into a data-driven segmentation model based on Point Transformer v3.

The objectives of this experimental evaluation are threefold:

- To assess the baseline performance of Point Transformer v3 on the ARCH dataset under standard training conditions.
- To quantify the contribution of the neuro-symbolic component by comparing the baseline model with its variant trained using LTNTorch.
- To analyze the impact of different logical rules and predicate formulations on the overall segmentation accuracy and rule satisfiability.

We first describe the experimental setup, including implementation details, training parameters, and dataset splits. We then report the results of both baseline and neuro-symbolic configurations, comprised of a detailed analysis of per-class performance and qualitative evaluations. Finally, we present ablation studies that examine the influence of individual rules and knowledge components.

The remainder of this chapter is organized as follows: Section 5.1 introduces the dataset used for evaluation; Section 5.2 details the experimental configuration; Section 5.3 reports the quantitative and qualitative results of the baseline model; Section 5.4 presents the quantitative and qualitative results of the neuro-symbolic configurations, along with ablation studies on the knowledge base.

5.1 Dataset

The experiments in this work are conducted on the ARCH dataset [5], a large-scale benchmark for semantic segmentation of 3D point clouds in architectural environments. The dataset comprises richly annotated scans of historical buildings, providing point-wise semantic labels for various architectural elements such as walls, floors, ceilings, door_windows, columns, and moldings. Each point is described by its spatial coordinates, color information, and surface normal vectors, offering a comprehensive representation of the 3D structure and appearance of the scenes.

The dataset is publicly available at `https://archdataset.polito.it/`. It is designed to support research in automated scene understanding, and digital heritage documentation. The diversity and complexity of the scenes in ARCH make it a suitable testbed for evaluating neuro-symbolic approaches that leverage both data-driven learning and domain-specific logical constraints.

5.2 Experimental setup

All experiments were conducted using configuration files, which define every component of the training and evaluation pipeline.

All the models were trained for 3000 epochs using the AdamW optimizer *lr = 0.002, weight decay = 0.005* with a OneCycleLR scheduler employing cosine annealing *pct_start=0.04, div_factor=10, final_div_factor=1000*. The default batch size was set to 12, with mixed-precision training enabled. A special learning rate of *0.0002* is used for the Block layers of encoder and decoder. In some experiments batch size was varied, in that case learning rate of each module was adjusted proportionally. The metric used to select the best model on the validation set was the mean Intersection-over-Union (mIoU).

The ARCH dataset was split into training, validation, and test partitions, following the same division as in [21],and all augmentations followed the same pipeline described in Section 4.2.

5.3 Baseline

In this section, the results obtained by Ptv3 when trained and tested on the reduced ARCH dataset are analyzed and compared with previously published benchmarks. Subsequently, the results obtained by Ptv3 when trained and tested on the complete dataset are examined, in order to identify a baseline configuration for the neuro-symbolic framework.

5.3.1 Comparison with Existing Benchmarks

To ensure a fair comparison with previously published results on the ARCH dataset, this experiment was conducted on a dataset obtained after removing points belonging to class 9 (Other), following the evaluation protocol of [21]. Metrics were computed over the remaining nine semantic classes for both official test scenes, `Test_SMV` and `Test_SMG`.

As mentioned in 4.2.2, during the prediction of the point clouds using the DGCNN architecture each scene was spatially divided into N parallelograms that were forwarded independently through the network, and predictions were produced only for the sampled points within each volume. Therefore, in [21], the evaluation metrics for DGCNN were computed only on the subsampled points that were directly predicted within each parallelogram, without projecting the results to the full-resolution point cloud. As a consequence, a conspicuous amount of points was excluded from the quantitative assessment. In contrast, our evaluation protocol follows the standard practice of reassigning predictions to every original point through nearest-neighbor interpolation, ensuring that all geometric details contribute to the reported metrics. This methodological difference should be taken into account when interpreting the magnitude of the performance gap.

Training protocol. Two experiments were conducted to compare the two configurations, differing from the inclusion of additional descriptors in input. Both variants were trained using identical hyperparameters and data augmentations, with batch size set to 6, differing only in the inclusion of the six handcrafted descriptors. For each setup, the best model according to validation performance was evaluated on the two test scenes. All metrics were computed on the full-resolution point clouds after re-projection.

Quantitative Analysis

Table 5.1 summarizes the results obtained by our baseline model, Point Transformer v3 (PTV3), and the benchmark DGCNN [21], both evaluated separately on the `SMV` and `SMG` test scenes.

We can see that PointTransformer v3 outperforms DGCNN by a substantial margin on the `SMV` test scene, achieving a mean IoU of 77.51% compared to 55.56% for DGCNN, representing an improvement of over 21 percentage points. This significant gain highlights the effectiveness of the transformer architecture in capturing complex spatial relationships and fine-grained geometric details inherent in architectural point clouds. Instead on the `SMG` test scene, PTV3 attains a mean IoU of 71.79%, surpassing DGCNN's 59.97% by nearly 12 percentage points. Although the overall accuracy and F1-score are slightly lower for PTV3 in this

Table 5.1: Comparison of segmentation performance on the ARCH dataset (class 9 *Other* excluded). Precision, recall, and F1 are weighted averages. The results for DGCNN are in [21] under the name of DGCNN-Mod+3Dfeat.

Scene	Method	Overall Acc.	Precision	Recall	F1	mIoU
SMV	DGCNN	86.46%	85.32%	86.46%	85.57%	55.56%
	Ptv3	91.02%	90.98%	91.02%	90.94%	77.51%
	Diff	**+4.56%**	**+5.66%**	**+4.56%**	**+5.37%**	**+21.95%**
SMG	DGCNN	91.44%	91.73%	91.45%	91.48%	59.97%
	Ptv3	90.07%	90.94%	90.07%	90.29%	71.79%
	Diff	-1.37%	-0.79%	-1.38%	-1.19%	**+11.82%**

scene, the substantial increase in mean IoU indicates that the model delivers more balanced performance across all classes, particularly benefiting underrepresented or geometrically complex categories. We will see in 5.3 that DGCNN outperforms PTV3 only in the 'floor' class.

In addition to the baseline configuration, PTV3 was also trained by incorporating the same handcrafted 3D geometric descriptors used in [21] (verticality, planarity, omnivariance, surface variation, and normalized height) as additional input channels. However, this extended setup resulted in minimal changes in performance, with variations within one percentage point for most metrics. This suggests that the transformer backbone effectively learns comparable geometric relationships directly from raw spatial coordinates and normal vectors, without requiring explicit 3D features.

Table 5.2 reports the per-class precision, recall, F1-score, and IoU obtained on the `SMV` test scene for both Point Transformer v3 and DGCNN [21].

Table 5.2: Per-class segmentation performance on the `SMV` test scene (class 9 *Other* excluded).

Class	**DGCNN-Mod (with additional 3D features)**				**Point Transformer v3 (no 3D features)**			
	Prec.	**Rec.**	**F1**	**IoU**	**Prec.**	**Rec.**	**F1**	**IoU**
arch (0)	0.2619	0.0631	0.1017	0.0536	**0.7585**	**0.6536**	**0.7022**	**0.5410**
column (1)	0.6940	0.6780	0.6859	0.5219	**0.8858**	**0.9397**	**0.9120**	**0.8382**
moldings (2)	0.5217	0.4418	0.4784	0.3144	**0.7247**	**0.7167**	**0.7207**	**0.5633**
floor (3)	0.7927	0.8921	0.8394	0.7233	**0.9365**	**0.9696**	**0.9527**	**0.9097**
door_window (4)	0.5660	0.2615	0.3578	0.2178	**0.8298**	**0.7206**	**0.7713**	**0.6278**
wall (5)	0.8447	0.8999	0.8714	0.7721	**0.9324**	**0.9350**	**0.9337**	**0.8756**
stairs (6)	0.8563	0.7837	0.8184	0.6926	**0.9261**	**0.8901**	**0.9077**	**0.8310**
vault (7)	0.8295	0.9474	0.8845	0.7929	**0.8875**	**0.9646**	**0.9244**	**0.8595**
roof (8)	0.9611	0.9464	0.9537	0.9115	**0.9853**	**0.9432**	**0.9638**	**0.9301**
Mean	0.7031	0.6571	0.6657	0.5556	**0.8741**	**0.8592**	**0.8654**	**0.7751**

Point Transformer v3 substantially improves the segmentation quality across all

architectural categories. Large gains are observed in highly variable or structurally fine-grained classes such as *arch* (F1: 0.10 → 0.70), *door_window* (0.36 → 0.77), and *moldings* (0.48 → 0.72), demonstrating the model's ability to capture both local and global spatial context without explicit handcrafted features. For highly regular classes such as *floor*, *wall*, and *roof*, PTV3 also achieves higher precision and IoU, confirming its robustness in large planar regions. Overall, the mean IoU increases from 0.56 for DGCNN to 0.78 for PTV3, marking a substantial performance improvement on the `SMV` test scene. Table 5.3 presents the per-class segmentation results obtained on the `SMG` test scene for both Point Transformer v3 and DGCNN [21].

Table 5.3: Per-class segmentation performance on the `SMG` test scene (class 9 *Other* excluded).

Class	**DGCNN (with additional 3D features)**				**Point Transformer v3 (ours, no 3D features)**			
	Prec.	**Rec.**	**F1**	**IoU**	**Prec.**	**Rec.**	**F1**	**IoU**
arch (0)	0.5318	0.2578	0.3472	0.2100	**0.6290**	**0.7341**	**0.6775**	**0.5123**
column (1)	0.8497	0.9250	0.8858	0.7949	**0.9825**	**0.9840**	**0.9832**	**0.9671**
moldings (2)	0.6502	0.5959	0.6219	0.4512	**0.8012**	**0.8207**	**0.8109**	**0.6819**
floor (3)	0.9566	0.9030	0.9290	0.8673	0.7958	**0.9078**	0.8481	0.7363
door_window (4)	0.1355	0.1956	0.1601	0.0870	**0.4217**	**0.6021**	**0.4960**	**0.3298**
wall (5)	0.8797	0.8551	0.8672	0.7655	**0.9396**	**0.9239**	**0.9317**	**0.8721**
stairs (6)	0.4661	0.7101	0.5628	0.3915	**0.6045**	**0.9155**	**0.7282**	**0.5726**
vault (7)	0.8909	0.9688	0.9282	0.8660	**0.9602**	**0.9894**	**0.9746**	**0.9504**
roof (8)	0.9753	0.9880	0.9816	0.9630	0.9825	0.8509	0.9120	0.8382
Mean	0.7040	0.7110	0.6982	0.5997	**0.7908**	**0.8587**	**0.8180**	**0.7179**

On the `SMG` test scene, we can notice that Point Transformer v3 again surpasses DGCNN in nearly all classes, confirming its robustness across different architectural contexts. The overall mean IoU rises from 0.60 to 0.72, accompanied by a higher macro F1-score (0.82 vs. 0.70). However, the pattern of improvements differs from the `SMV` scene, reflecting the distinct spatial composition and material variability of `SMG`.

Significant gains are observed in classes with irregular geometry or lower representation in the dataset, such as *arch* (F1: 0.35 → 0.68), *door_window* (0.16 → 0.50), and *stairs* (0.56 → 0.73). These improvements indicate that the transformer architecture generalizes better to spatially complex elements compared to DGCNN. Consistent improvements also occur in structural classes with strong geometric regularity, such as *column*, *wall*, and *vault*, where PTV3 achieves near-perfect F1-scores (above 0.93) and IoU values exceeding 0.86. The *roof* and *floor* classes, on the other hand, show smaller or mixed changes: PTV3 achieves lower recall for the *roof* class (0.85 vs. 0.99), and reduced precision for *floor*.

Qualitative Analysis

Test Scene Sacri Monti of Varallo (SMV) Figure 5.1 presents a qualitative comparison of semantic segmentation results on the `SMV` test scene, contrasting DGCNN and Point Transformer v3 (PTv3) including all versions. The prediction of each model is rendered with a consistent color scheme.

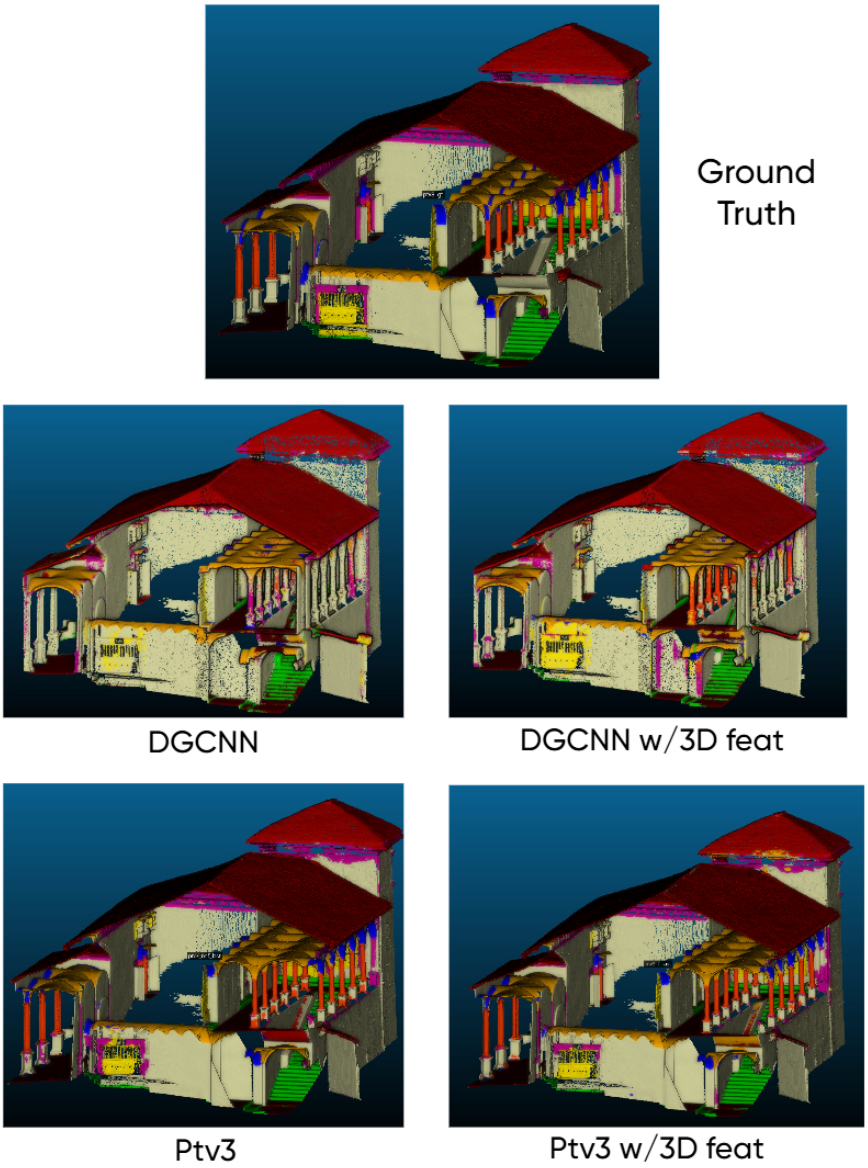


Figure 5.1: Existing Benchmarks: Qualitative comparison test scene SMV It depicts ground truth, DGCNN, PTv3, and their respective variants trained with handcrafted 3D geometric features.

The ground truth scene (top) exhibits a high level of geometric complexity, characterized by arch elements, thin columns, and staircases. DGCNN predictions capture the general spatial layout but suffer from irregular boundaries and partial misclassifications around the *arches*, *columns*, *door/windows* and *moldings*. The inclusion of 3D geometric descriptors moderately enhances these areas especially by improving continuity along the *columns* and better capturing the *door/window* regions; however, the global structure remains coarse.

By contrast, PTv3 produces substantially cleaner and more continuous segmentation throughout all classes. A clear improvement can be seen in classes such as arch, columns and moldings. The variant trained with 3D features exhibits mixed results, for example further refinement in some small-scale details—such as consistent labeling of the column bases, or misclassifications of moldings near door_windows, without improving the overall result. The model trained without additional 3D features already provides robust and visually coherent predictions.

In summary, the `SMV` scene confirms the trends observed in the quantitative analysis: Point Transformer v3 consistently outperforms DGCNN, achieving higher structural fidelity and reduced fragmentation, while handcrafted features mainly contribute minor refinements in fine-grained areas rather than broad improvements.

Scene Sacri Monti of Ghiffa (SMG) Figure 5.2 presents a qualitative comparison of segmentation outputs produced by DGCNN and Point Transformer v3 on representative regions from the `SMG` test scene. The visualizations highlight the main differences in segmentation quality and boundary delineation for this scene.

Visually, DGCNN struggles to delineate fine structural elements such as *arches*, *moldings* and *columns*, often producing discontinuous or noisy class boundaries. In this case the addition of 3D features to DGCNN enhances boundary definition and improves recognition of *columns* regions, while *arches* and *moldings* remain poorly segmented.

In contrast, PTv3 yields markedly cleaner and more coherent predictions, successfully recovering the global architectural layout and preserving structural continuity of columns, arches and moldings. The segmentation of this model outperforms all classes, apart from floor and roof. The *column* elements are consistently segmented, while *arches* have a good recall (0.73) with low precision (0.63) resulting in coarse predictions. The inclusion of handcrafted 3D descriptors have mixed results, improving the classification for some classes and worsening for others i.e. improving arch and moldings segmentation while worsening door window's. The observed drop in precision for the *floor* class, exhibited by Ptv3 on `SMG` can be fully explained by a particular portion of the cloud, the roof on the left, where the misclassification of the entire roof as floor led to a major drop in class performances. In fact we have lower precision but slightly higher recall for floor class.

Overall, the qualitative inspection corroborates the quantitative results presented

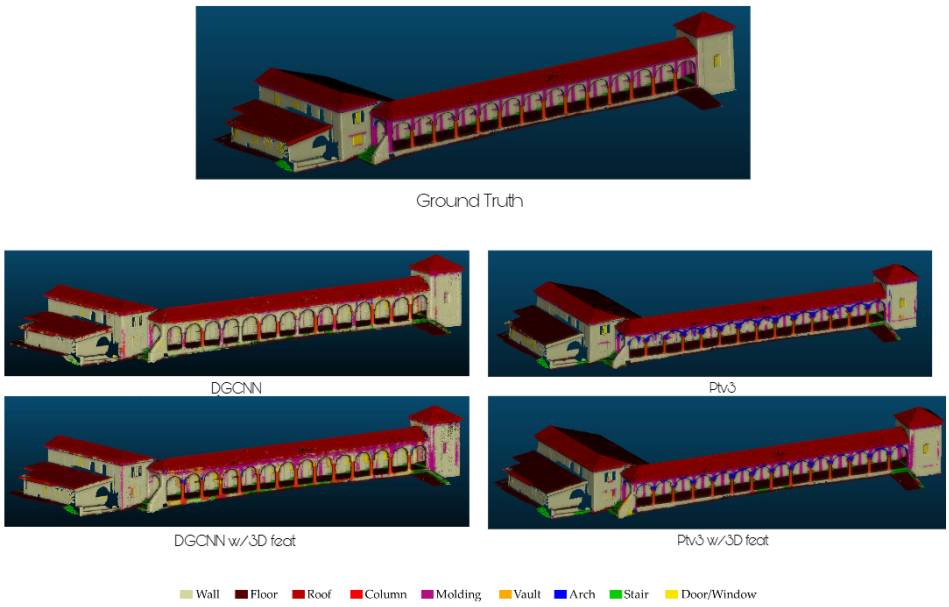


Figure 5.2: Existing Benchmarks: Qualitative comparison test scene SMG
It depicts ground truth, DGCNN, PTv3, and their respective variants trained with handcrafted 3D features.

above. PT v3 produces more coherent and geometrically consistent segmentation, whereas DGCNN exhibits higher fragmentation. The additional geometric descriptors have marginal results but are not essential for achieving semantically accurate and visually stable predictions.

Summary

In summary, PTV3 exhibits a consistent advantage over DGCNN across all semantic categories, with the largest relative improvements concentrated in fine-grained and topologically challenging elements. This trend highlights the ability of transformer-based encoders to adapt to diverse spatial contexts and scene structures without relying on handcrafted 3D descriptors.

5.3.2 Baseline Analysis: Full ARCH Dataset

In this section results from the complete Arch Dataset, that contains an additional class, are presented and analyzed. The models are trained from scratch with the new dataset and tested again. The main objective of this experiment is to establish a robust baseline on the complete dataset, which will serve as a reference point for

subsequent evaluations of the neuro-symbolic learning framework.

A comparison assessing the difference in behavior of the Ptv3 architecture on the reduced and complete dataset can be found in the end of this section: 5.3.2.

To obtain a more comprehensive and statistically stable evaluation of the baseline performance, the two official test scenes (`SMV` and `SMG`) were merged into a single aggregated test split. This configuration allowed to assess the overall behavior of the model on the entire ARCH dataset and to isolate the effect of incorporating handcrafted 3D geometric descriptors such as verticality, planarity, omnivariance, surface variation, and normalized height. These features were concatenated into the standard feature vector (`rgb + normals`) to form the "3D-feature" configuration, while the ""No 3D-features"" baseline variant relied solely on raw geometric and color information.

Training protocol. Six experiments were conducted to compare the two configurations (with and without additional 3D features) with different batch sizes. Both variants were trained using identical hyperparameters and data augmentations, differing only in the inclusion of the six handcrafted descriptors. Batch size was varied between 3, 6, and 12 to explore scaling effects, and for each setup, the best model according to validation performance was evaluated on the aggregated test set. All metrics were computed on the full-resolution point clouds after re-projection. The metrics were then averaged throughout experiments of the same configuration.

Quantitative Analysis

Table 5.4 shows average and max on the metrics across different training configurations reporting both validation and test results in each setting.

Table 5.4: Results on the full ARCH dataset (SMV + SMG) comparing the baseline Point Transformer v3 with and without handcrafted 3D features. Metrics are calculated for each experiment on the weights of the epoch that performed best in validation.

Setting	**No 3D features**			**With additional 3D features**		
	allAcc	mAcc	mIoU	allAcc	mAcc	mIoU
Test	0.852	**0.768**	0.651	**0.854**	0.762	0.651
Validation	0.686	0.567	0.432	**0.786**	**0.731**	**0.596**
Max Test	0.855	**0.797**	**0.669**	**0.863**	0.764	0.656
Max Validation	0.687	0.571	0.447	**0.796**	**0.743**	**0.605**

The results indicate a strong consistency between the two variants in the test

set. Across all metrics, the inclusion of handcrafted geometric descriptors produced marginal differences — within 1 percentage points in the Test set. Interestingly, the validation set displayed a larger gap between configurations (mIoU 0.43 vs. 0.59), suggesting that handcrafted features improve generalization under domain shift or reduced scene coverage.

Per-class behavior. A more granular comparison of per-class IoU and accuracy (Figure 5.3) in the test and validation set shows a comparison between both configurations in both Validation and Test. While Figure 5.4 shows see the relative representation of each class in the training, validation and test split.

In general the classes with the worst performance in the test split are *arch*, *moldings*, *door_window* and *other*. These classes are more complex and variegated in their geometry, making them more challenging to segment accurately. Results on the Validation split highlight the strong dependency of validation performance on the inclusion of additional geometric descriptors. The most pronounced gains occur in *arch*, *moldings*, and *vault*, where IoU values nearly double when using handcrafted features. This effect suggests that the handcrafted descriptors provide meaningful geometric regularization in smaller subsets of the dataset. In the validation split, for the configuration using additional geometric descriptors, the per-class performance follows the same trend observed in the test set, with the lowest segmentation quality occurring for : *arch*, *moldings*, *door_window*, and *other*. This further confirms the challenges associated with these categories. In addition, the class *stairs* also shows low performance in the validation split, but it can be explained by the ambiguous labeling of ramp structures in the dataset.

It can be said that the representation of the classes in the training split is not the solely reason for model performance during test and validation, highlighting the model capacity to learn simple patterns from few samples.

In the next subsection a qualitative analysis will be performed, focusing on each scene in the Validation and Test split for a total of three scenes. To do so a representative model was chosen for each configuration, in particular both of them were trained with batch size 6 and unvaried hyper parameters.

A table with the metrics achieved by both models can be found in 5.5

Qualitative analysis

This analysis will focus on three scenes, the first from the validation split and the other two from the test split. Predictions are shown in A.1, A.2 and A.3. The goal of the analysis is to highlight the strenght and the weaknesses of both variant. In the validation scene A.1 we can see an overall improvement in the predictions of the model trained including additional descriptors, resulting in cleaner and more

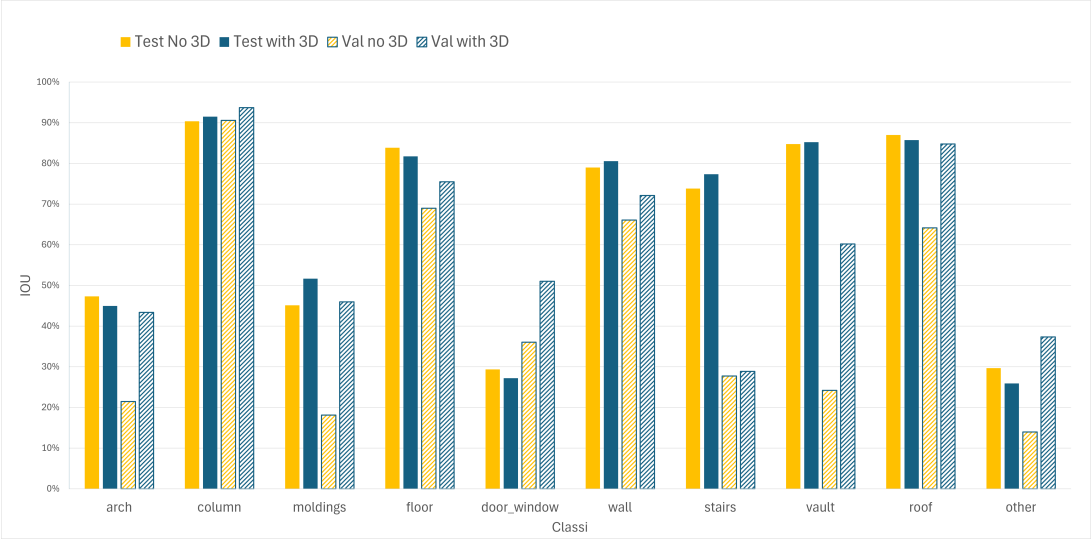


Figure 5.3: Per-class IoU comparison between Ptv3 models on the full arch dataset

Trained with (blue) and without (orange) handcrafted 3D features. Solid bars indicate aggregated test results (SMV+SMG), while hatched bars correspond to the validation scene.

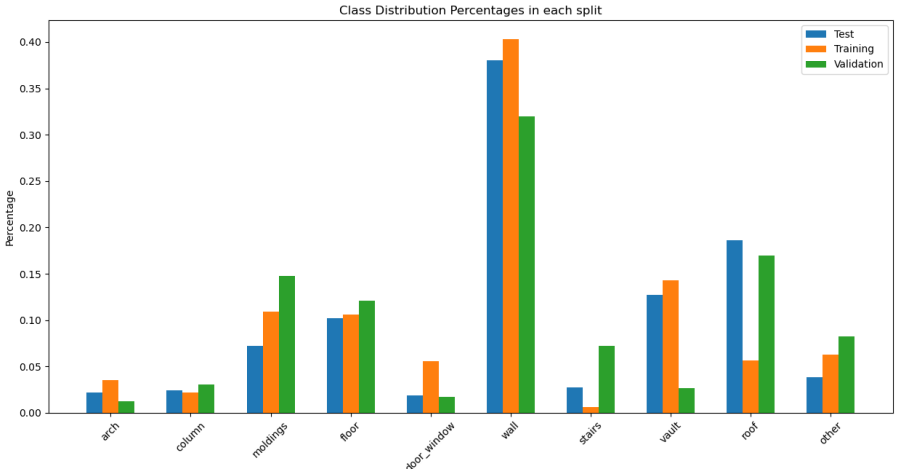


Figure 5.4: Training label distribution

The chart shows the percentage distribution of each class in the splits of the ARCH dataset. Interesting to notice is the high class imbalance, with some classes like *wall* and *vault* being significantly more represented than others like *stairs* and *columns*.

Validation	no feat bs6	3d feat bs6	$\Delta_{3dfeat-nofeat}$	**Test**	no feat bs6	3d feat bs6	$\Delta_{3dfeat-nofeat}$
allAcc	68.48%	78.19%	**9.70%**		85.33%	86.27%	**0.94%**
mAcc	56.74%	72.92%	**16.17%**		79.65%	76.04%	-3.61%
mIoU	44.73%	60.52%	**15.79%**		66.87%	65.59%	-1.28%
Per Class IoU							
arch	20.52%	47.72%	**27.20%**		45.18%	39.43%	-5.75%
column	89.03%	94.90%	**5.87%**		92.38%	92.74%	**0.36%**
moldings	20.84%	43.19%	**22.36%**		45.68%	55.47%	**9.79%**
floor	68.39%	75.99%	**7.60%**		87.16%	80.40%	-6.76%
door_window	44.70%	56.50%	**11.80%**		37.32%	30.73%	-6.59%
wall	67.76%	70.87%	**3.12%**		77.59%	83.69%	**6.10%**
stairs	27.86%	28.49%	**0.63%**		75.91%	77.39%	**1.48%**
vault	26.34%	62.02%	**35.68%**		88.19%	85.14%	-3.05%
roof	65.67%	85.13%	**19.46%**		89.70%	85.48%	-4.22%
other	11.45%	36.29%	**24.83%**		29.62%	26.55%	-3.07%

Table 5.5: Comparison of the baseline with and without the addition of descriptors in input

correct boundaries. This is confirmed by the metrics. Regarding the test scene SMV A.2 we can see that the model including additional 3D features struggles in the correct predictions of arches (b), while the other variant offers a more consistent recognition. Notably the part of the element arch that is badly recognized is the one with low verticality. Another typical error of the model including the descriptors is the loss of consistency in the prediction of the roof edges (c). In (a) instead we can see an example where the model including descriptors outperforms the variant, in particular in the tower on the left, we can see how the model correctly predicted a downpipe (even if the ground truth lacks this annotation) and separated it from the molding, while the variant predicted non-consistently. Finally, as we can see in (c) the model with descriptors have a more coherent predictions of moldings and door_window, as shown by the class IoU.

Regarding the SMG test scene A.3 we still see misclassification of roof structures in both variants (a), while the model including the descriptors misclassifies some irregular part of the floor (left side of a). In (b) its noticeable the improvement of the variant including descriptors in predicting downpipes, being part of others, with little misclassifications, while the variant struggles. The problem in predicting arch structures for the model that includes descriptors remains, as can be seen in (d). In (c) we can notice that both models could not predict the paintings in the corridor, being part of the *other* class; this can be explained by the scarcity of this type of element in the training split.

Summary

The results presented in this section establish a solid baseline for subsequent neuro-symbolic experiments. Point Transformer v3 demonstrates strong generalization and high segmentation accuracy across both validation and test scenes, confirming its capability to learn complex spatial relationships directly from raw point attributes. The inclusion of hand-crafted geometric descriptors—such as verticality, omnivariance, planarity, and surface variation—was shown to exert only a marginal influence on quantitative performance, particularly in large and diverse test scenarios. Nonetheless, these descriptors improved model stability and generalization on the smaller validation subset by providing additional low-level geometric priors. Since these geometric quantities facilitate the encoding of spatial relationships relevant for logical reasoning, the configuration including handcrafted 3D features was retained for all subsequent neuro-symbolic experiments. This choice ensures that both the neural and logical components operate on a shared, geometrically enriched representation of the scene, facilitating the integration of domain-specific reasoning within the LTNTorch framework.

The baseline analysis also highlighted certain semantic categories—such as *arch*, *moldings*, *door/window*, and *other*—that remain challenging for purely data-driven segmentation approaches. These classes exhibit lower IoU and precision scores, likely due to the amount of different shapes and styles they can assume and/or limited representation in the training set. Addressing these challenges motivates the exploration of neuro-symbolic methods that can leverage domain knowledge and logical constraints to enhance segmentation performance in these difficult categories.

Comparison Reduced and Full Dataset

A comparison was performed between the model trained and tested on the reduced dataset and the model trained and tested on the complete dataset. Both configurations were trained using the same hyper-parameters, including a batch size of 6. In this comparison the versions trained without additional 3D descriptors were selected.

Table 5.6 shows that Ptv3 performs worse on the complete dataset, with a decrease of -5.67% mIoU on SMV and -12.23% on SMG.

Table 5.7 shows that the segmentation of class other in the SMV scene performed well, achieving an IoU of nearly 57%. In contrast, classes such as arch, moldings, wall and vault experienced decreases of about 2–10%, and door_window dropped by almost 20%. The only category that improved clearly in the complete dataset was column, with an increase of approximately 6%.

Table 5.8 shows that the class other achieves only 9% IoU in SMG. Most semantic classes show consistent decreases: arch drops by 25% IoU, door_window by 20%,

	Method	Overall Acc.	Precision	Recall	F1	mIoU
SMV	Reduced	91.02%	90.98%	91.02%	90.94%	77.51%
	Complete	88.29%	88.62%	88.29%	88.10%	71.84%
	Diff	**-2.73%**	**-2.36%**	**-2.73%**	**-2.84%**	**-5.67%**
SMG	Reduced	90.07%	90.94%	90.07%	90.29%	71.79%
	Complete	84.58%	85.95%	84.58%	85.05%	59.55%
	Diff	**-5.49%**	**-4.99%**	**-5.49%**	**-5.24%**	**-12.23%**

Table 5.6: Comparison of Ptv3 results in reduced and complete Arch dataset

SMV	reduced				complete			
	Prec.	Rec.	F1	IoU	Prec.	Rec.	F1	IoU
arch (0)	0.7585	**0.6536**	**0.7022**	**0.541**	**0.7935**	0.5606	0.657	0.4893
column (1)	0.8858	**0.9397**	0.912	0.8382	**0.9808**	0.91	**0.944**	**0.894**
moldings (2)	**0.7247**	**0.7167**	**0.7207**	**0.5633**	0.623	0.6668	0.6442	0.4751
floor (3)	**0.9365**	0.9696	**0.9527**	**0.9097**	0.9165	**0.9716**	0.9432	0.8926
door_window (4)	0.8298	**0.7206**	**0.7713**	**0.6278**	**0.8689**	0.4766	0.6156	0.4447
wall (5)	**0.9324**	**0.935**	**0.9337**	**0.8756**	0.9203	0.919	0.9197	0.8513
stairs (6)	0.9261	0.8901	0.9077	0.831	**0.9368**	**0.8946**	**0.9152**	**0.8437**
vault (7)	**0.8875**	0.9646	**0.9244**	**0.8595**	0.8229	**0.9793**	0.8943	0.8089
roof (8)	0.9853	**0.9432**	**0.9638**	**0.9301**	**0.9874**	0.9259	0.9557	0.9151
other(9)					**0.7151**	**0.7367**	**0.7257**	**0.5695**
Mean	**0.8741**	**0.8592**	**0.8654**	**0.7751**	0.8565	0.8041	0.8215	0.7184

Table 5.7: Per class comparison of Ptv3 results, scene SMV, when trained and tested on reduced or complete Arch dataset

and wall, moldings, vault by around 5%.

The large discrepancy in the performance of the *other* class between the two scenes can be explained by their different compositions: SMV consists mostly of balustrades, which the model predicts easily, while SMG contains more complex elements such as shutters, paintings, and chimneys.

In summary, across both scenes, there is a consistent decrease in metrics for classes such as moldings, vault, wall, and roof, while arch and door_window experience the most significant drops. Overall, it is clear that the complete dataset introduces greater noise, making correct predictions much more difficult for the model. This dataset may be better suited for neuro-symbolic learning, where regularization and ad hoc rules can improve training stability.

SMG	reduced				complete			
	Prec.	Rec.	F1	IoU	Prec.	Rec.	F1	IoU
arch (0)	**0.629**	**0.7341**	**0.6775**	**0.5123**	**0.5554**	0.3294	0.4135	0.2607
column (1)	0.9825	**0.984**	**0.9832**	**0.9671**	**0.9882**	0.9729	**0.9805**	**0.9617**
moldings (2)	**0.8012**	**0.8207**	**0.8109**	**0.6819**	0.7536	**0.8216**	0.7861	0.6476
floor (3)	**0.7958**	**0.9078**	**0.8481**	**0.7363**	**0.8781**	**0.8475**	**0.8625**	**0.7583**
door_window (4)	**0.4217**	**0.6021**	**0.496**	**0.3298**	**0.1667**	0.4235	0.2392	0.1359
wall (5)	**0.9396**	**0.9239**	**0.9317**	**0.8721**	0.9102	0.9005	0.9053	0.8271
stairs (6)	**0.6045**	**0.9155**	**0.7282**	**0.5726**	**0.6199**	**0.8512**	**0.7173**	**0.5592**
vault (7)	**0.9602**	0.9894	**0.9746**	**0.9504**	0.9071	**0.9948**	0.9489	0.9028
roof (8)	**0.9825**	**0.8509**	**0.912**	**0.8382**	**0.9498**	0.8442	0.8939	0.8081
other(9)					**0.1623**	**0.1813**	**0.1713**	**0.0937**
Mean	**0.7908**	**0.8587**	**0.8180**	**0.7179**	0.6891	0.7167	0.6919	0.5955

Table 5.8: Per class comparison of Ptv3 results, scene SMG, when trained and tested on reduced or complete Arch dataset

5.4 Experiment with knowledge base

In this section, we present the experiments conducted to evaluate the impact of progressively integrating logical knowledge into the segmentation pipeline. Starting from a baseline model trained solely on data-driven supervision (described in detail in Section 5.3.2), we introduce different sets of neuro-symbolic rules and analyze how each configuration influences performance. To avoid biased choices when selecting which rules to retain, all intermediate evaluations are carried out exclusively on the **validation split**. Based on these validation results, a subset of rule configurations is then chosen, the model is trained accordingly, and final performance is assessed on the **held-out test split**.

Training protocol. Various experiments were conducted to compare the rule configurations with batch size set to 6. All variants, including the baseline were trained with the inclusion of additional 3D descriptors, using identical hyperparameters and data augmentations, apart from the weight given to the domain knowledge loss, which was sometimes varied from the default value of 1. For each setup, the best model according to validation performance was evaluated on the aggregated test set. All metrics were computed on the full-resolution point clouds after re-projection.

By examining the behavior of the system across increasingly rich knowledge bases, we aim to understand how logical constraints interact with the learned features, how they modify the model's predictions, and to what extent they contribute to

improvements in scene-level semantic segmentation. A first series of experiments is showed in Table 5.9

	RFC	ADD1	ADD2	ADD3	FWV	MR	CR	AR	DWS	DWP
1	✓									
2	✓									
3	✓		✓							
4	✓				✓					
5	✓		✓		✓					
6	✓	✓			✓					
7	✓	✓								
8	✓		✓		✓					
9	✓			✓	✓					
10	✓			✓						
11								✓		
12							✓			
13									✓	
14										✓
15						✓				

Table 5.9: Series of experiments linked to the sets of rules included in the training process. The definition of each set can be found in Section 4.2.3.

To investigate how logical priors can help stabilize broad structural categories such as roof and floor, we introduce a series of rules designed to enforce geometric and contextual consistency. These categories often act as global scene anchors, yet they remain challenging in certain scenarios, as pointed out in Section 5.3.2. Each experiment applies increasingly rich variants of these rules (e.g., RFC, RFC+ADD, RFC+ADD+FWV), allowing us to isolate the contribution of each individual constraint set.

As documented in Table 5.10, introducing this family of rules leads to a slight improvement in the validation split, averaging a +1.7% increase in overall accuracy and +0.67% in mean intersection over union. Classes that are consistently improved include *moldings*, *wall*, *stairs*, *vault*, *roof*, and *other*, while some categories show a consistent decrease, namely *arch*, *column*, *floor*, and *door_window*. Regarding the contribution of the singular sets, we observed that the rules *ADD1*, *ADD2*, and *ADD3* were ineffective, whereas adding the *FWV* rule consistently resulted in a slight increase in mIoU.

In addition to these structural rules, ad hoc rules for certain classes were created, namely *AR*, *CR*, *DWS*, *DWP*, *MR*. Each experiment applied only one of them at a time to asses the contribution of each set.

Validation	Baseline	Mean Exp.	Δ	Representative Δ
allAcc	77.65%	79.37%	**1.72%**	**2.51%**
mIoU	60.11%	60.78%	**0.67%**	**1.85%**
iou arch	47.72%	40.87%	-6.85%	-6.13%
iou column	94.90%	92.52%	-2.38%	-0.42%
iou moldings	43.19%	48.67%	**5.47%**	**10.15%**
iou floor	75.99%	75.00%	-0.99%	-1.61%
iou door_window	56.50%	53.81%	-2.69%	-0.75%
iou wall	70.87%	73.29%	**2.41%**	**2.42%**
iou stairs	28.49%	30.91%	**2.42%**	**0.69%**
iou vault	62.02%	63.89%	**1.86%**	**2.94%**
iou roof	85.13%	88.47%	**3.35%**	**4.46%**
iou other	36.29%	40.37%	**4.08%**	**6.71%**

Table 5.10: Mean performance and improvement (Δ) over the baseline 5.3.2 for experiments introducing combinations of Roof–Floor Consistency (RFC) and Floor–Wall Verticality (FWV) rules.

Validation	BASELINE	AR	CR	DWS	DWP	MR
allAcc	77.65%	1.26%	2.67%	2.08%	-0.34%	-1.49%
mIoU	60.11%	0.31%	1.74%	-0.43%	-2.57%	-1.43%
iou arch	47.72%	-11.99%	-2.67%	-8.77%	-7.34%	-7.39%
iou column	94.90%	-0.37%	-4.21%	-0.63%	-0.84%	-0.51%
iou moldings	43.19%	8.11%	12.06%	8.55%	1.51%	-6.83%
iou floor	75.99%	-0.34%	-2.52%	0.43%	-2.11%	-0.04%
iou door_window	56.50%	1.32%	-4.35%	-19.89%	-21.85%	-2.33%
iou wall	70.87%	1.07%	4.31%	1.66%	-1.11%	-3.46%
iou stairs	28.49%	5.21%	4.19%	18.55%	2.59%	6.54%
iou vault	62.02%	-1.47%	-2.57%	0.46%	8.59%	4.17%
iou roof	85.13%	2.85%	1.22%	3.05%	1.26%	1.35%
iou other	36.29%	-1.27%	11.90%	-7.76%	-6.43%	-5.76%

Table 5.11: Mean improvement over the baseline 5.3.2 for experiments introducing per class rules

We can see in Table 5.11 that, in all cases, rules intended to improve a specific semantic class instead led the model to predict that class less frequently. As discussed in 4.2.3, this behavior is expected when using rules of the form

$$\mathit{Class}(x) \implies \mathit{Characteristic}(x),$$

since the model can satisfy the rule by negating the antecedent rather than enforcing the consequent. In any case, some of the implemented rules—such as AR, CR, and DWS—still improve the general results, particularly for classes like *moldings*, *stairs*, and *roof*. Notably, CR also improves the class *other*, suggesting that the additional geometric and contextual constraints introduced by this rules may help the model better distinguish residual or ambiguous regions.

Based on the results of the previous experiments, we selected a subset of rules that either showed a positive impact on the validation split or could be meaningfully extended due to their logical relevance. This selection was guided by the aggregated trends reported in Table 5.10 for the structural rules and Table 5.11 for the per-class rules. This analysis allowed us to discard rule configurations that consistently degraded performance while retaining those that provided stable or meaningful improvements.

The retained rule sets were then combined to produce several new model configurations (listed in Table 5.12), which were subsequently evaluated in both the validation split and the held-out test split to assess the generalization capacity of the resulting knowledge bases.

	RFC+ERFC	ADD2	ADD3	FWV+EFWV	MR+EMR	CR+ECR
FINAL1	✓	✓	✓	✓	✓	✓
FINAL2	✓			✓	✓	✓
FINAL3	✓			✓		

Table 5.12: Rule configurations used in the final experiments.

The results of the evaluation on these configurations, together with a selected set of strong performers, are presented in Table 5.13.

Validation	Final1	Final2	Final3	l=0.1 RFC	RFC_ADD1_FWV	l=2 RFC_ADD1_FWV	RFC_ADD3_FWV	CR
allAcc	**1.10%**	-0.20%	**0.22%**	**2.83%**	**2.51%**	**1.41%**	**1.41%**	**2.67%**
mIoU	**0.08%**	-1.15%	-0.45%	**1.60%**	**1.85%**	**0.54%**	**0.47%**	**1.74%**
Test	Final1	Final2	Final3	l=0.1 RFC	RFC_ADD1_FWV	l=2 RFC_ADD1_FWV	RFC_ADD3_FWV	CR
allAcc	-0.73%	-0.36%	-0.64%	**0.31%**	**0.29%**	**0.54%**	**0.44%**	-1.60%
mIoU	-0.22%	**0.23%**	**0.52%**	**1.06%**	**0.73%**	**1.93%**	**1.96%**	-1.43%

Table 5.13: Improvement over the baseline for Validation and Test Split on the model trained with best performing configurations

We can see that the chosen configuration produced mixed results, without clearly improving over the baseline. Instead, minimal sets of rules—especially those focusing on the most represented classes, *floor*, *roof*, *other*, and *wall*—consistently improved performance. In contrast, rules targeting single classes proved ineffective in the test set.

Focusing on the two configurations that demonstrated steady improvements across both splits and achieved the best performance in the validation and test sets—namely *RFC, ADD3, FWV* and *RFC, lambda=0.1*—their corresponding results are presented in Table 5.14 and Table 5.15.

	arch	column	moldings	floor	door_window	wall	stairs	vault	roof	other
Test	**16.77%**	**0.24%**	-3.37%	-2.35%	-4.88%	-0.81%	**1.49%**	**2.42%**	**0.33%**	**0.78%**
Validation	-5.34%	-1.06%	**8.48%**	-1.85%	-5.48%	**3.50%**	**3.27%**	**1.57%**	**4.07%**	**8.82%**

Table 5.14: Improvement in IoU per class for configuration that performed best on the validation set, containing set of rules *RFC l=0.1*

	arch (0)	column (1)	moldings (2)	floor (3)	door_window (4)	wall (5)	stairs (6)	vault (7)	roof (8)	other (9)
Test	**16.81%**	-0.42%	-1.99%	**2.98%**	-0.30%	-1.25%	-2.50%	**5.04%**	-0.24%	**1.49%**
Validation	-4.84%	-2.52%	**3.82%**	-1.06%	-5.46%	**2.73%**	**2.13%**	**3.08%**	**1.67%**	**5.10%**

Table 5.15: Improvement in IoU per class for configuration that performed best on the test set, containing set of rules *RFC, ADD3, FWV*

In both configurations, the pattern of improvement differs between the test and validation splits, particularly for the classes *arch*, *moldings*, and *wall*. This suggests that the distribution of these classes varies between the test and validation scenes—especially for the *arch* class, where the difference in improvement exceeds 20% IoU. It should also be noted that the baseline configuration used for comparison was the worst-performing one for arches among all baseline setups (Figure 5.3), with the best baseline achieving an IoU approximately 10% higher. Even accounting for this, we can still assert that the configuration incorporating rules had a positive effect on arch segmentation. Overall, the regularization introduced by the analyzed rules enhances the segmentation of the *arch* class in the test set, while also improving *vault*, *other* and *roof* or *floor* in both splits.

We can also note that the configuration containing the set of rules *FWV* improves the metrics for *floor* compared to the other configuration and the baseline, showing a clear enhancement attributable to the regularization introduced by this set.

Figures B.1, B.2, and B.3 present visualizations of the predictions from the baseline and the two best-performing configurations using rules.

In the validation scene (Fig. B.1), we observe less "bleeding" in the predictions of *floor*, *roof*, and *stairs* (b), a more consistent classification of *vault* (c), and overall better classification of *moldings*.

In the test scene SMV (Fig. B.2), we see reduced confusion in the prediction of roof edges (a, b), improved segmentation of *arch* (b), and a case where the configuration *RFC l=0.1* (c) fails to predict a door that was partially detected by the baseline. Both rule-based configurations are also less precise than the baseline in segmenting *moldings*.

The last analyzed scene is SMG (Fig. B.3), where the error regarding the roof on the left is not fully corrected by any configuration, even though the segmentation pattern changes—for example, by restricting the prediction of *other* (bottom left) or partially predicting some areas correctly as *roof* (bottom right). Image (b) also shows an improvement in the segmentation of *arch* in this scene.

Chapter 6

Conclusion

The work presented in this thesis provide a comprehensive evaluation of the effects of integrating logical knowledge into a transformer-based segmentation pipeline for point cloud semantic segmentation, specifically on the *ARCH* dataset. A key finding of this work is that enlarging the knowledge base by adding many heterogeneous rules does not necessarily lead to better performance. In fact, broad or overly specific rule sets often introduced noise into the optimization process, leading to inconsistent or even degraded results across different scenes.

Conversely, a slim and carefully selected subset of rules proved consistently beneficial. Rules targeting highly represented and structurally defining classes through geometric cues such as coplanarity—particularly for *roof*, *floor*, and *other*—provided a form of regularization that improved not only their own predictions but also those of related classes. For instance, although the rules were primarily designed for major structural elements, improvements were observed in classes such as *arch*, and *vault*, as documented in Tables 5.14 and 5.15. This suggests that enforcing geometric and contextual consistency on core architectural components indirectly stabilizes predictions in neighboring or semantically connected classes.

The results also show that rule sets designed for individual classes and that do not introduce new geometric cues rarely produced the intended effect. Only a few per-class rules, such as *CR* or *FWV*, yielded measurable improvements, indicating that this kind of semantic constraints must be formulated with particular care.

Overall, these findings highlight that within a neuro-symbolic framework, *the quality, focus, and geometric grounding of the knowledge base matter more than its size*. Compact rule sets that leverage meaningful geometric cues and align with the architectural structure and dataset distribution provide a stable and effective means of guiding the learning process.

6.1 Future Work

The promising results obtained with compact structural rule sets open several directions for future research. First, rules defined at the *instance level* could more accurately capture the geometric structure of indoor and heritage environments. Second, rules derived from new *rich geometric insights*, such as the co-planarity prior used in this work, may provide stronger and more expressive constraints than point-wise rules alone.

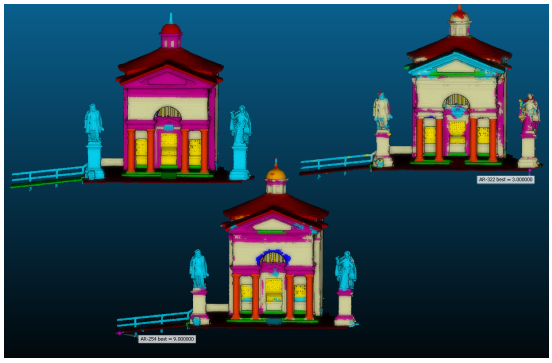
6.2 Thesis Contributions

This thesis presented the first complete analysis of a new transformer architecture for point cloud semantic segmentation on the *ARCH* dataset, achieving a new state-of-the-art on its benchmark. It introduced a full neuro-symbolic learning pipeline built directly on top of the Pointcept engine [4] and extended using LTNTorch [26], integrating Real Logic with a modern point-cloud transformer. The work included a substantial effort in rule design, implementation, and systematic experimental evaluation, resulting in actionable insights into how logical priors can interact with geometric features to guide 3D semantic segmentation.

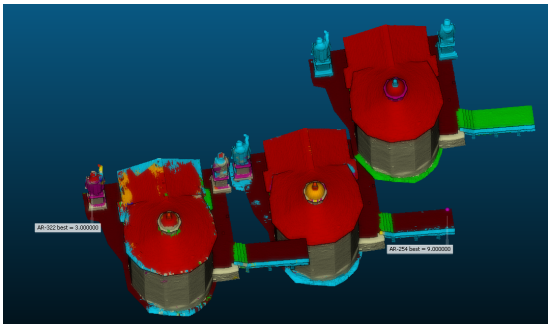
Together, these contributions demonstrate both the potential and the limitations of neuro-symbolic methods in large-scale 3D understanding, setting the foundations for more expressive and reliable regularization-based neuro-symbolic systems in future work.

Appendix A

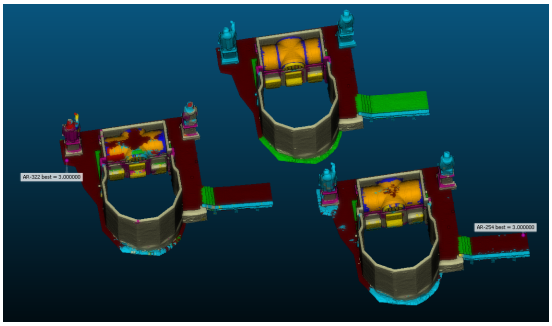
Baseline comparison



(a) 3d features bottom, no 3d features top

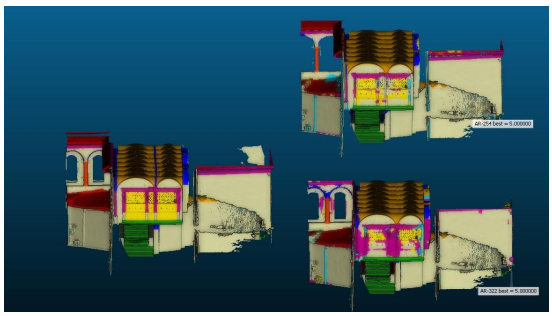


(b) 3d features center, no 3d features left

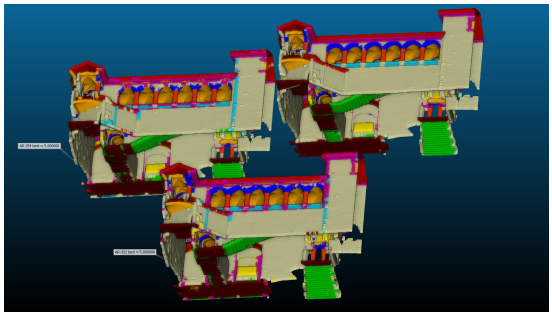


(c) 3d features right, no 3d features left

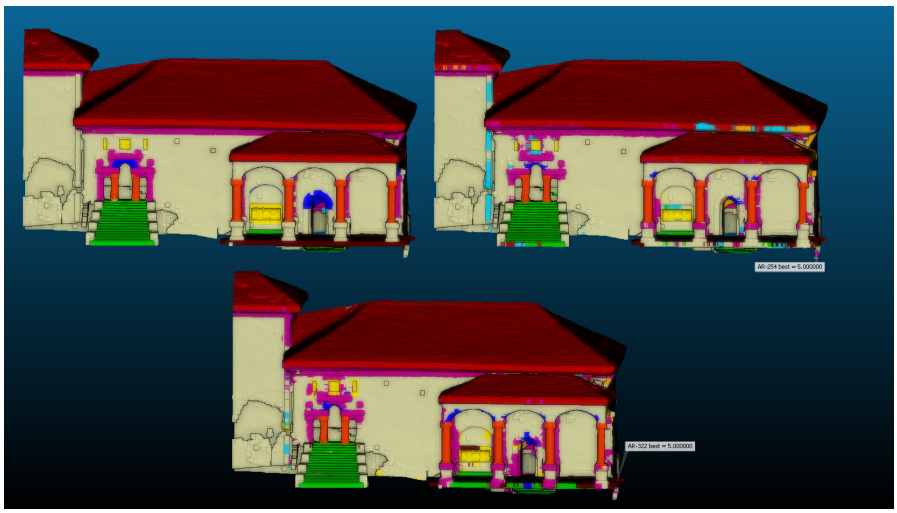
Figure A.1: Baseline variants: Qualitative comparison on validation scene. Each subfigure shows a different viewpoint comparing the ground truth, model trained without 3D features , and model trained with 3D features.



(c)

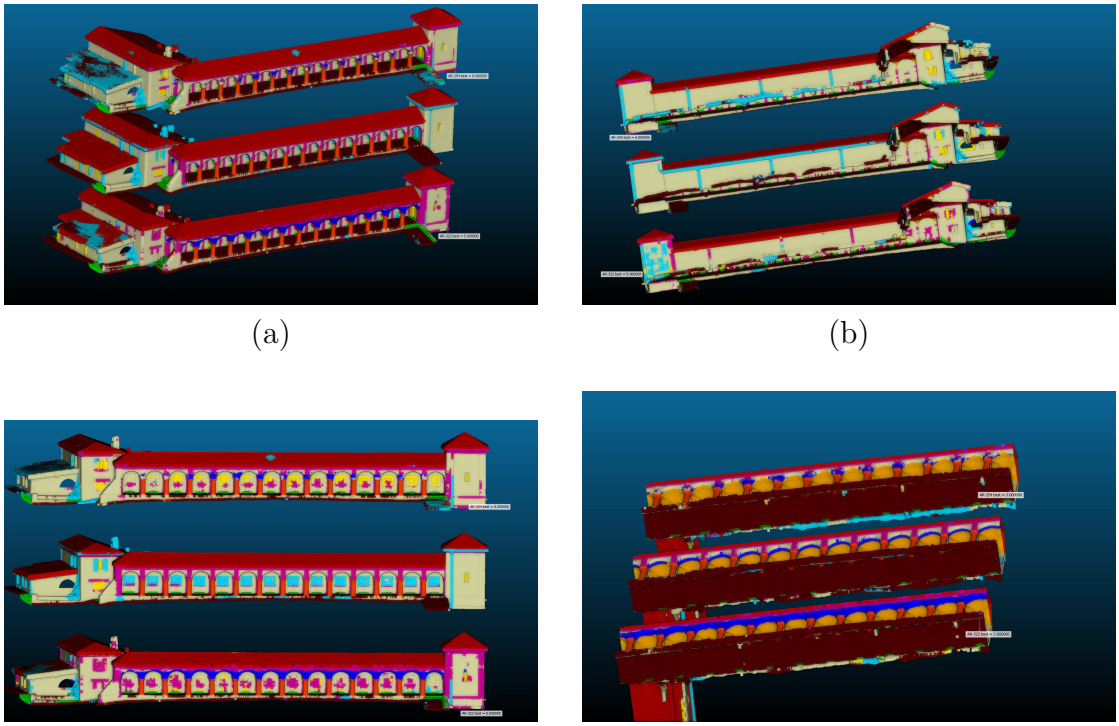


(b)



(a)

Figure A.2: Baseline variants: Qualitative comparison on SMV scene. Each view shows predictions from the model trained without 3D features (bottom), with 3D features (top), and the ground truth.



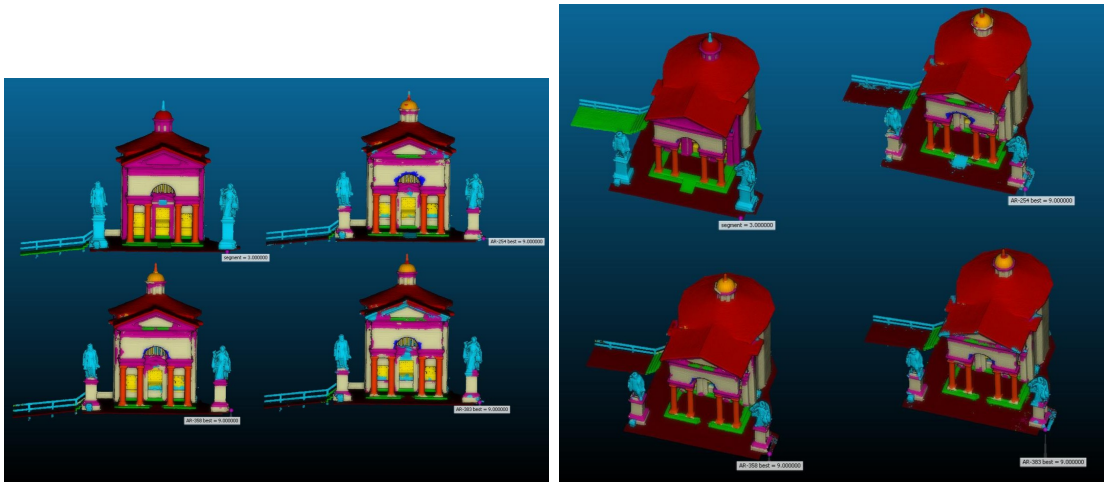
(a) (b)

(c) (d)

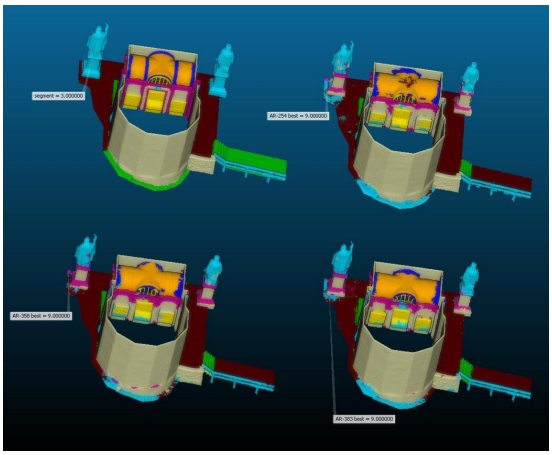
Figure A.3: Baseline variants: Qualitative comparison on SMG scene. Each view shows three corresponding segmentations: from the model trained without 3D features (bottom), with 3D features (top), and the ground truth.

Appendix B

Rules effect comparison

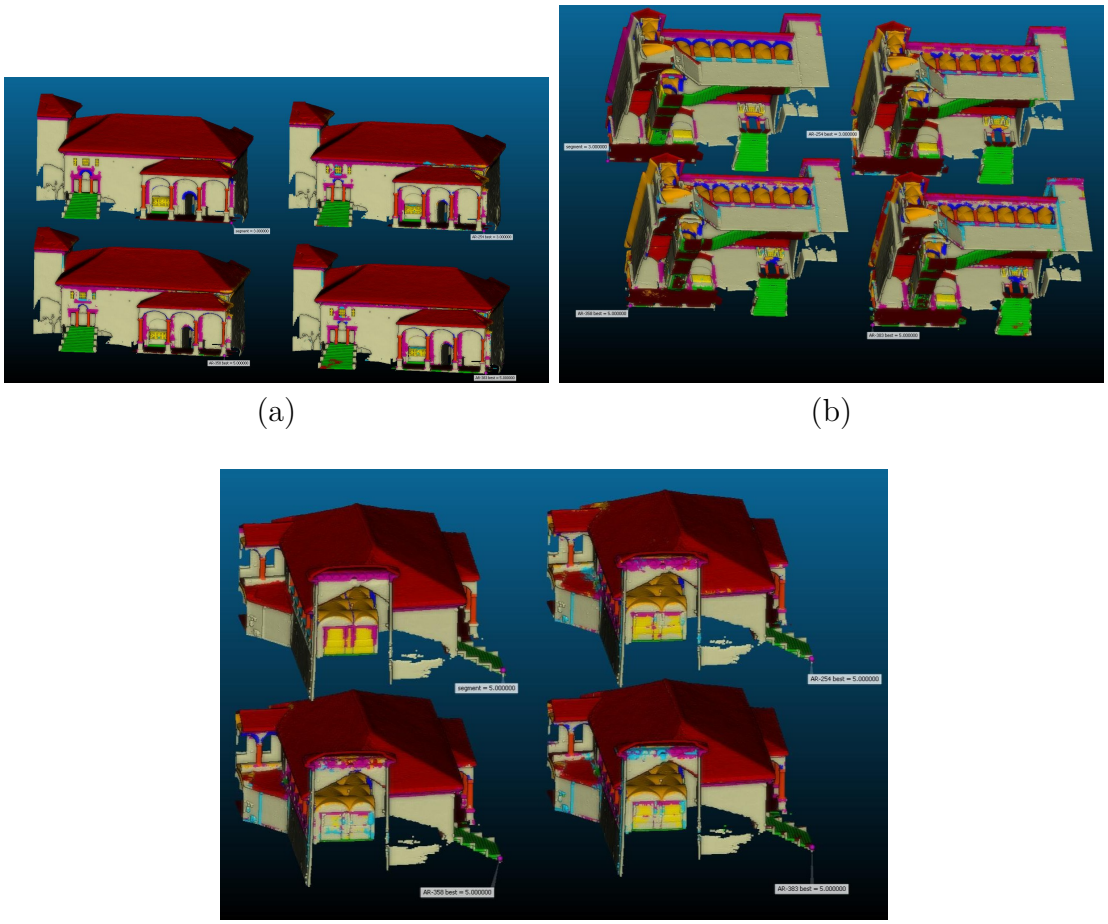


(a) (b)



(c)

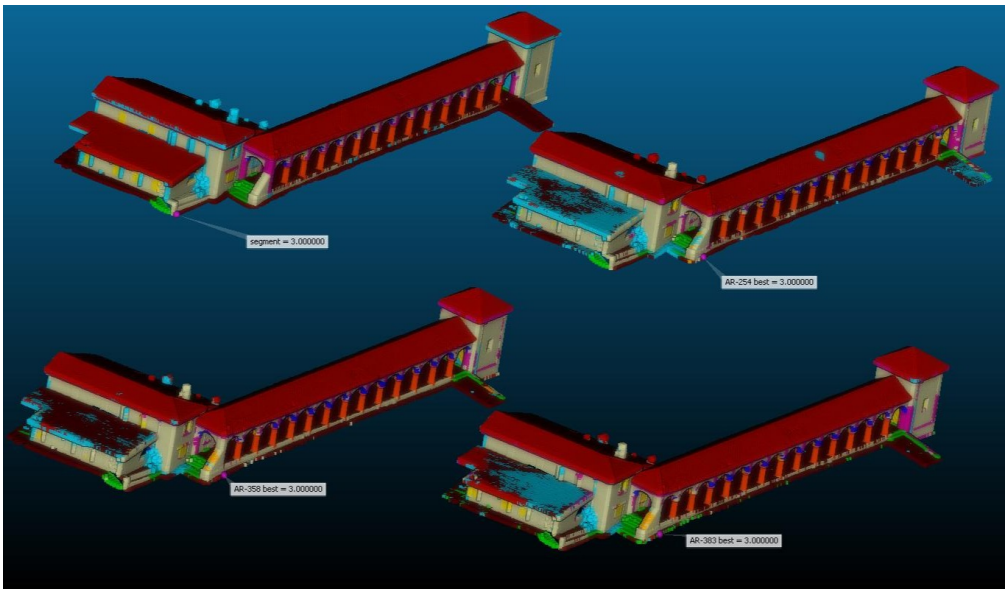
Figure B.1: Qualitative comparison rules against baseline
Each subfigure shows a different viewpoint comparing the ground truth (top left), baseline (top right) , configuration with rules *RFC, ADD3, FWV* (bottom right) and configuration with rules *RFC l=0.1* (bottom left)



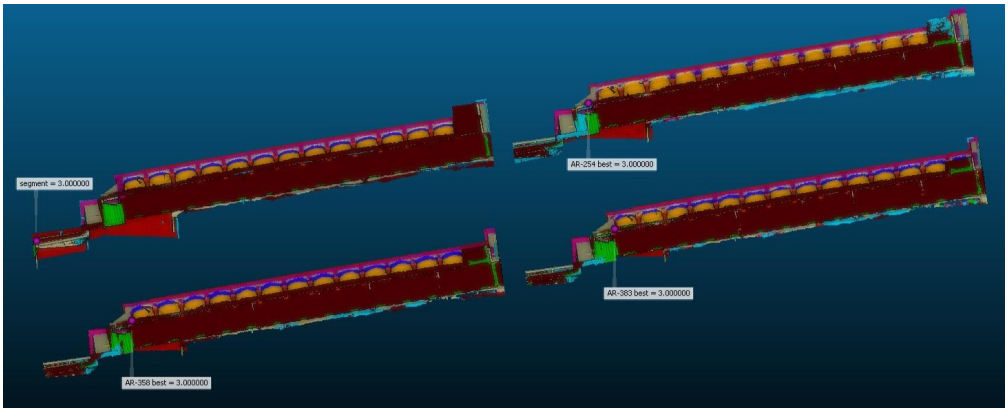
(a) (b)

(c)

Figure B.2: Qualitative comparison rules against baseline
Each subfigure shows a different viewpoint comparing the ground truth (top left), baseline (top right) , configuration with rules *RFC, ADD3, FWV* (bottom right) and configuration with rules *RFC l=0.1* (bottom left)



(a)



(b)

Figure B.3: Qualitative comparison rules against baseline
Each subfigure shows a different viewpoint comparing the ground truth (top left), baseline (top right) , configuration with rules *RFC, ADD3, FWV* (bottom right) and configuration with rules *RFC l=0.1* (bottom left)

Bibliography

[1] Brandon C. Colelough and William Regli. *Neuro-Symbolic AI in 2024: A Systematic Review*. 2025. arXiv: 2501.05435 [cs.AI]. URL: https://arxiv.org/abs/2501.05435 (cit. on p. 1).

[2] Xiaoyang Wu, Li Jiang, Peng-Shuai Wang, Zhijian Liu, Xihui Liu, Yu Qiao, Wanli Ouyang, Tong He, and Hengshuang Zhao. «Point Transformer V3: Simpler Faster Stronger». In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. June 2024, pp. 4840–4851 (cit. on pp. 1, 11, 17, 21).

[3] Samy Badreddine, Artur S. d'Avila Garcez, Luciano Serafini, and Michael Spranger. «Logic Tensor Networks». In: *CoRR* abs/2012.13635 (2020). arXiv: 2012.13635. URL: https://arxiv.org/abs/2012.13635 (cit. on p. 1).

[4] Pointcept. *Pointcept*. https://github.com/Pointcept/Pointcept. GitHub repository. 2025 (cit. on pp. 2, 17, 28, 50).

[5] F. Matrone, A. Lingua, R. Pierdicca, E. S. Malinverni, M. Paolanti, E. Grilli, F. Remondino, A. Murtiyoso, and T. Landes. «A BENCHMARK FOR LARGE-SCALE HERITAGE POINT CLOUD SEMANTIC SEGMENTATION». In: *The International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences* XLIII-B2-2020 (2020), pp. 1419–1426. DOI: 10.5194/isprs-archives-XLIII-B2-2020-1419-2020. URL: https://isprs-archives.copernicus.org/articles/XLIII-B2-2020/1419/2020/ (cit. on pp. 2, 17, 30).

[6] Henry Kautz. «The third AI summer: AAAI Robert S. Engelmore Memorial Lecture». In: *AI Magazine* 43 (Mar. 2022), pp. 105–125. DOI: 10.1002/aaai.12036 (cit. on p. 5).

[7] Luciano Serafini and Artur S. d'Avila Garcez. «Logic Tensor Networks: Deep Learning and Logical Reasoning from Data and Knowledge». In: *CoRR* abs/1606.04422 (2016). arXiv: 1606.04422. URL: http://arxiv.org/abs/1606.04422 (cit. on pp. 5–7, 12).

[8] Yong He, Hongshan Yu, Xiaoyan Liu, Zhengeng Yang, Wei Sun, Yaonan Wang, Qiang Fu, Yanmei Zou, and Ajmal Mian. «Deep Learning based 3D Segmentation: A Survey». In: *CoRR* abs/2103.05423 (2021). arXiv: 2103.05423. URL: https://arxiv.org/abs/2103.05423 (cit. on pp. 11, 14).

[9] Yanni Ma, Yulan Guo, Hao Liu, Yinjie Lei, and Gongjian Wen. «Global Context Reasoning for Semantic Segmentation of 3D Point Clouds». In: Mar. 2020, pp. 2920–2929. DOI: 10.1109/WACV45572.2020.9093411 (cit. on p. 11).

[10] Zhuyang Xie, Junzhou Chen, and Bo Peng. «Point Clouds Learning with Attention-based Graph Convolution Networks». In: *CoRR* abs/1905.13445 (2019). arXiv: 1905.13445. URL: http://arxiv.org/abs/1905.13445 (cit. on p. 11).

[11] Xiaoyang Wu, Yixing Lao, Li Jiang, Xihui Liu, and Hengshuang Zhao. *Point Transformer V2: Grouped Vector Attention and Partition-based Pooling*. 2022. arXiv: 2210.05666 [cs.CV]. URL: https://arxiv.org/abs/2210.05666 (cit. on p. 11).

[12] Xin Lai, Jianhui Liu, Li Jiang, Liwei Wang, Hengshuang Zhao, Shu Liu, Xiaojuan Qi, and Jiaya Jia. *Stratified Transformer for 3D Point Cloud Segmentation*. 2022. arXiv: 2203.14508 [cs.CV]. URL: https://arxiv.org/abs/2203.14508 (cit. on p. 11).

[13] Hengshuang Zhao, Li Jiang, Jiaya Jia, Philip Torr, and Vladlen Koltun. *Point Transformer*. 2021. arXiv: 2012.09164 [cs.CV]. URL: https://arxiv.org/abs/2012.09164 (cit. on p. 11).

[14] Dongran Yu, Bo Yang, Dayou Liu, Hui Wang, and Shirui Pan. *A Survey on Neural-symbolic Learning Systems*. 2023. arXiv: 2111.08164 [cs.LG]. URL: https://arxiv.org/abs/2111.08164 (cit. on p. 12).

[15] Michelangelo Diligenti, Marco Gori, and Claudio Saccà. «Semantic-based regularization for learning and inference». In: *Artificial Intelligence* 244 (2017). Combining Constraint Solving with Mining and Learning, pp. 143–165. ISSN: 0004-3702. DOI: https://doi.org/10.1016/j.artint.2015.08.011. URL: https://www.sciencedirect.com/science/article/pii/S0004370215001344 (cit. on p. 12).

[16] G. Forestier, C. Wemmert, and A. Puissant. «Coastal image interpretation using background knowledge and semantics». In: *Computers and Geosciences* 54 (2013), pp. 88–96. ISSN: 0098-3004. DOI: https://doi.org/10.1016/j.cageo.2012.11.023. URL: https://www.sciencedirect.com/science/article/pii/S0098300412004062 (cit. on pp. 12, 13).

[17] Eleonora Grilli, Alessandro Daniele, Maarten Bassier, Fabio Remondino, and Luciano Serafini. «Knowledge Enhanced Neural Networks for Point Cloud Semantic Segmentation». In: *Remote Sensing* 15 (May 2023), p. 2590. DOI: 10.3390/rs15102590 (cit. on p. 12).

[18] Ivan Donadello, Luciano Serafini, and Artur d'Avila Garcez. *Logic Tensor Networks for Semantic Image Interpretation*. 2017. arXiv: 1705.08968 [cs.AI]. URL: https://arxiv.org/abs/1705.08968 (cit. on p. 13).

[19] B. Wagner and A. S. d'Avila Garcez. «Neural-symbolic integration for fairness in AI». In: *AAAI 2021 Spring Symposium on Combining Machine Learning and Knowledge Engineering (AAAI-MAKE 2021)*. Vol. 2846. © 2021 Copyright for this paper by its authors. Use permitted under Creative Commons License Attribution 4.0 International (CC BY 4.0). 2021. URL: http://ceur-ws.org/Vol-2846/paper5.pdf (cit. on p. 13).

[20] Luca Bergamin, Giovanna Maria Dimitri, and Fabio Aiolli. *Integrating Background Knowledge in Medical Semantic Segmentation with Logic Tensor Networks*. 2025. arXiv: 2509.22399 [cs.CV]. URL: https://arxiv.org/abs/2509.22399 (cit. on p. 13).

[21] Francesca Matrone, Eleonora Grilli, Massimo Martini, Marina Paolanti, Roberto Pierdicca, and Fabio Remondino. «Comparing Machine and Deep Learning Methods for Large 3D Heritage Semantic Segmentation». In: *ISPRS International Journal of Geo-Information* 9.9 (2020). ISSN: 2220-9964. DOI: 10.3390/ijgi9090535. URL: https://www.mdpi.com/2220-9964/9/9/535 (cit. on pp. 14, 17, 21, 30–33).

[22] Yue Wang, Yongbin Sun, Ziwei Liu, Sanjay E. Sarma, Michael M. Bronstein, and Justin M. Solomon. *Dynamic Graph CNN for Learning on Point Clouds*. 2019. arXiv: 1801.07829 [cs.CV]. URL: https://arxiv.org/abs/1801.07829 (cit. on p. 14).

[23] Martin Weinmann, Boris Jutzi, Clément Mallet, and Michael Weinmann. «Geometric Features and Their Relevance for 3D Point Cloud Classification». In: *ISPRS Annals of the Photogrammetry, Remote Sensing and Spatial Information Sciences* IV-1/W1 (June 2017). DOI: 10.5194/isprs-annals-IV-1-W1-157-2017 (cit. on p. 17).

[24] Eleonora Grilli, Elisa Farella, Alessandro Torresani, and Fabio Remondino. «GEOMETRIC FEATURES ANALYSIS FOR THE CLASSIFICATION OF CULTURAL HERITAGE POINT CLOUDS». In: *ISPRS - International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences* XLII-2/W15 (Aug. 2019), pp. 541–548. DOI: 10.5194/isprs-archives-XLII-2-W15-541-2019 (cit. on p. 17).

[25] Maxim Berman and Matthew B. Blaschko. «Optimization of the Jaccard index for image segmentation with the Lovász hinge». In: *CoRR* abs/1705.08790 (2017). arXiv: 1705.08790. URL: http://arxiv.org/abs/1705.08790 (cit. on p. 20).

[26] Tommaso Carraro, Luciano Serafini, and Fabio Aiolli. *LTNtorch: PyTorch Implementation of Logic Tensor Networks*. 2024. arXiv: 2409.16045 [cs.AI]. URL: https://arxiv.org/abs/2409.16045 (cit. on pp. 29, 50).