JOT
JOURNAL OF
OBJECT TECHNOLOGY

Journal of Object Technology | **RESEARCH ARTICLE**

Structural-Semantic Clustering for Architectural Models

Thi Dinh Tran*, Maria Teresa Rossi*, Davide Soldati†, Mauro Sonzogni†, Amleto Di Salle*, Ludovico Iovino*, and Leonardo Mariani†

*Gran Sasso Science Institute, Italy
†University of Milano-Bicocca, Italy

ABSTRACT Architectural models are rich and useful representations of systems' designs, which can be reused and adapted to design new systems. The many publicly available architectural models (e.g., on GitHub) offer the opportunity to reuse domain knowledge to bootstrap the design of new systems, avoiding common mistakes and promoting the reuse of good principles and practices. Discovering related architectures to distill domain knowledge, however, is challenging, and this body of public knowledge often remains underexploited. To address this need, this paper investigates the application of structural-semantic clustering strategies to automatically cluster sets of architectural models. We consider multiple structural model clustering strategies, which represent models as graphs, and multiple semantic model clustering strategies, utilizing embeddings to compare the names and contents of models. To assess the considered strategies, we built a curated dataset of 1,202 manually clustered models collected from GitHub, which is the largest ground truth of clustered architectural models, to the best of our knowledge. The proposed empirical study thoroughly analyses the effectiveness of clustering strategies, since they are the instrument that can enable the discovery of implicit architectural knowledge available in models.

KEYWORDS Software Architecture, Architectural Models, Clustering, Structural Similarity, Semantic Similarity

1. Introduction

Architectural models are first-class artifacts in model-driven development, where key design and development decisions are made and represented through models (Feiler & Gluch 2012; Feiler & Rugina 2007). Although architectural models can facilitate development in many ways (Blouin & Borde 2020), such as enabling automatic code generation (Hatcliff et al. 2021), supporting test generation (Hatcliff et al. 2023), and providing formal verification and analysis (Stewart et al. 2021), they are also cumbersome and expensive to produce, often representing a barrier to adopting model-driven paradigms (Akthar et al. 2024).

Architectural models encode domain knowledge and reflect technical decisions made to address design problems; however, these decisions are *not unique* to a given system. This is evident in the diffusion of reference architectures, which provide architectural models ready to be reused by analysts to tackle well-known and common problems, such as the design of Industry 4.0 systems (Nakagawa et al. 2021), automotive systems (Behere & Törngren 2016), avionics (Tokar 2017), and robotics (Hochgeschwender et al. 2018). Most architectural problems can be efficiently solved by adapting solutions (i.e., models) previously designed by analysts who addressed similar problems in other contexts.

Discovering and exploiting the body of knowledge encoded in the many publicly available models remains a significant challenge. Although repositories host a large number of models, little work has focused on how to meaningfully aggregate them. The absence of such aggregations makes the systematic reuse of knowledge far more difficult than it should be. As a result, analysts miss opportunities to spot reusable architectural solutions, uncover recurring design patterns, or derive reference models within a domain (Losavio et al. 2015; Guth et al. 2016, 2017).

Some approaches investigated how to classify architectural models using pre-trained machine learning models (Babur 2019; Nguyen et al. 2019, 2021; López & Cuadrado 2020; López et al. 2022; Khalilipour et al. 2022). While these techniques may effectively group models within certain areas, they require ex-

JOT reference format:
Thi Dinh Tran, Maria Teresa Rossi, Davide Soldati, Mauro Sonzogni, Amleto Di Salle, Ludovico Iovino, and Leonardo Mariani. *Structural-Semantic Clustering for Architectural Models*. Journal of Object Technology. Vol. 25, No. 3, 2026.
http://dx.doi.org/10.5381/jot.2026.25.3.a24

pensive data collection and training phases to be applied. The need for labeled data is also a barrier to the extensive adoption of these approaches. Clustering approaches have been investigated as an appealing alternative to the lack of labeled data. Some works focused on the structural information to cluster models (Babur & Cleophas 2017; Clarisó & Cabot 2018), while some other on the semantic one (López et al. 2023; Rubei et al. 2021). When applied to architectural models, these perspectives are individually insufficient. Structural-only approaches capture topological similarity but fail to distinguish models with similar architectures and different design intents, as commonly observed in architectural models across distinct domains (Şahin et al. 2021). Semantic-only approaches, instead, are sensitive to inconsistent naming conventions and may overlook architectural context. The combination of semantic and structural information has been explored in the context of metamodels (Basciani et al. 2015, 2016), but not yet extensively experienced in the context of architectural models, which are the artifacts carrying the fine-grained knowledge about design decisions.

To address this gap, we propose investigating *structural–semantic model clustering* techniques that integrate two complementary perspectives: the structural perspective, in which models are represented as graphs and analyzed based on their architectural topology, and the semantic perspective, in which the textual elements of the models are represented as sets of keywords and compared using word-embedding techniques. Both perspectives are relevant to identifying closely related architectural models.

The output of the clustering is a dendrogram (Lukasová 1979) that analysts can use to choose at which level of abstraction the architectural models must be aggregated. Intuitively, clustering models at a high level of abstraction (i.e., creating few but large clusters) corresponds to aggregating the models that belong to the same *application domain* (e.g., automotive). While clustering models at a low level of abstraction (i.e., creating many small clusters) corresponds to aggregating models designed to solve the same *specific problem* in a similar way (e.g., models of IoT systems for controlling the temperature in a room).

We assessed model clustering by mining *models and clusters from GitHub data*. In particular, we extracted a dataset of 1,202 architectural models in the AADL (Architecture Analysis and Design Language) (Feiler et al. 2006) format from GitHub. We manually analyzed all the extracted models and built a curated ground truth of clusters with models aggregated according to two criteria: models referring to the same application domain and models designed to solve the same specific problem. To the best of our knowledge, this dataset, along with the ground truth we have built, represents the largest repository of architectural models extracted from the wild. We have made it publicly available as part of the materials for reproducing our study. To date, clustering methods specifically tailored to AADL architectural models remain largely unexplored, and no standard baselines have been established for this context. Therefore, our evaluation adopts a systematic comparative analysis of structural, semantic, and hybrid clustering strategies. We make our contributions and the full replication package publicly available for third-party inspection and reuse (Tran et al. 2025).

2. AADL Architectures

As models that can be clustered to distill knowledge about how similar problems have been addressed from an architectural perspective, we focused on AADL models (Feiler & Gluch 2012). We selected this type of model because, according to this study (Malavolta et al. 2012), the three top-ranked architectural languages used in industry are AADL, ArchiMate (Josey et al. 2016), and industry-specific notations. Recent studies continue to exploit AADL for architecture-level assurance of safety-critical systems, including formal verification extensions and automated analysis workflows, demonstrating its ongoing relevance in current research (Xu et al. 2025; Zou et al. 2025; Hatcliff et al. 2025). Moreover, GitHub contains an impressive number of AADL specifications, providing a basis for the proposed study. To assess whether such artifacts remain actively maintained, we conducted a longitudinal GitHub analysis using the official Search API. Between January 2025 and February 2026, *6–25 public AADL-related repositories per month* received push updates while explicitly referencing AADL in their metadata, suggesting sustained activity beyond 2025. In total, *127 distinct repositories*, containing *3727 AADL models*, were updated throughout 2025. [1]

AADL targets real-time, safety-critical embedded systems where specific validation and verification capabilities are crucial to demonstrate correctness across all dimensions of the system (Mkaouar et al. 2020). AADL consists of a textual and graphical language with execution semantics for modelling the architecture and its target platforms. AADL uses the XMI interchange format to persist AADL models and ensure their interoperability with other tools.

AADL can model embedded systems as component-based system architecture, with interactions represented as flows, service calls, and shared access (Yu et al. 2013). AADL can deal with task execution and communication with precise timing semantics, can represent execution platforms and specify bindings, operational modes, and fault-tolerant configurations.

AADL models can have a hierarchical organization, where each component has a type, may have sub-components, may extend or refine others, and may include valued properties (Hugues & Brau 2014). The interaction between components is modelled as connectors, where the features are connection points.

Figure 1 shows an example of a home automation system named `Model-of-Smart-Home-System`, with the concrete syntax specification (textual) on top and the corresponding graphical representation on the bottom. The System called HOME contains an Arduino that binds a server and a router connected to the internet. The Arduino includes a proximity sensor and a temperature sensor, accessible by exploring its subcomponents, and regulates a door sensor and an actuator to open or close the door. The server accepts commands from a user, stores them, and forwards the requests to the Arduino, which replies to the user.

[1] Counts were obtained via monthly GitHub Search API queries using `aadl in:name,description,readme archived:false is:public pushed:YYYY-MM-DD..YYYY-MM-DD`.

```
package iot::home::integration
public
  system router
  features
      internal_eth : requires bus access iot::common::platform::ethernet;
      external_int : requires bus access iot::common::platform::internet;
  end router;
  system home
  features
      external_net    : requires bus access iot::common::platform::internet;
      user_request    : in event data port iot::common::datatypes::request;
      user_reply      : out event data port iot::common::datatypes::reply;
  flows
      flow_request         : flow sink user_request;
      flow_reply           : flow source user_reply;
  end home;
  system implementation home.i
  subcomponents
      internal_net : bus iot::common::platform::ethernet;
      server       : system iot::home::server::integration::server.i;
      arduino      : system iot::sap_node::integration::sap_node.i;
      router       : system router;
```

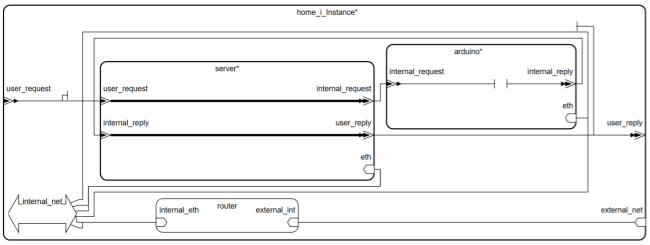

Figure 1 AADL spec and diagram of an IoT architecture

3. Structural-Semantic Clustering

Structural-Semantic Clustering (SSC) exploits two main elements to identify clusters of models: the structure and the semantic content of the models. Figure 2 shows an overview of the approach, which consists of three main steps.

In the first step, *Converting Architectural Models*, SSC extracts the relevant information from the architectures that must be compared. This mainly consists of two representations: a *graph-based representation* of the models, to study the similarity of the models according to their structure, and a *semantic representation* of the models, to study the similarity of the models according to their goal, represented by the set of labels that appear within the models.

In the second step, *Calculating Similarity Matrix*, SSC computes, for each pair of models to be compared, both a *structural similarity index*, derived from their graph-based representations, and a *semantic similarity index*, derived from their name and text-based representation. These indices are combined into a *composed similarity index* that is used to populate a *similarity matrix*, which is the matrix used to produce the actual clusters.

In the third step, *Clustering*, SSC produces a *dendrogram* with many different possible aggregations, representing *alternative abstraction levels* that can be used to aggregate the models. The analysts, based on their preference, can decide the cut level that best describes the intended result.

3.1. Converting Architectural Models

This step extracts both a graph-based and a semantic representation of the information contained in AADL models.

Regarding the *graph-based representation*, SSC derives a directional graph from each model, where the nodes are the components and the edges represent functional and behavioral relationships between the components. This allows us to capture not only the hierarchical and structural arrangement of the components, but also the functional interactions that characterize the dynamic behavior of the system.

Specifically, given a set of labels L, SSC uses a labeled directed graph $G = (N, E, l_N, l_E)$ to represent the structure of a model, where N is a set of nodes, $E \subseteq N \times N$ is a set of edges, $l_N : N \to L$ and $l_E : E \to L$ are the functions associating a label with each node and with each edge, respectively. Given an AADL model *aadl*, SSC defines the graph as follows:

- each component in *aadl* is a distinct node $n \in N$
- the type t of a component n in *aadl* is translated into a label, such that, $l_N(n) = t$
- each relation of type *hierarchical*, *feature* and *general connection* occurring from component $c_1 \in aadl$ (mapped to $n_1 \in N$) to component $c_2 \in aadl$ (mapped to $n_2 \in N$) is mapped into an edge $e \in E = (n_1, n_2)$, with $l_L(e) \in \{h, f, gc\}$, representing relations of type *hierarchical*, *feature* and *general connection*, respectively.

The graph-based representation intentionally abstracts from information that is not essential for understanding the components and their relations. In particular, it does not include external references, which link entities not available in the analyzed architecture, user-defined names (e.g., the name of the component and the name of the attributes), which do not carry structural information, and system operation modes, which refer to architecture instances and not architectures. This design choice reduces noise and variability in structural matching, enabling a more robust comparison of how components are organized and interact. At the same time, the information excluded from the structural representation is not discarded, but is instead considered by the semantic component of our approach.

This notion of graphs induces a comparison strategy among graphs. That is, given a graph G_1 and a graph G_2, $G_1 = G_2$ iff there is a bijection $map : N_1 \to N_2$, s.t.

- node labels match: $l_{N_1}(n_i) = l_{N_2}(f(n_i))$
- edges match: $(n_i, n_j) \in E_1$ iff $(f(n_i), f(n_j)) \in E_2$
- edge label match: $l_{E_1}(n_i, n_j) = l_{E_2}(f(n_i), f(n_j))$

Semantic representation combines two complementary sources of information: (i) the *model name representation*, and (ii) the *text-based representation*. The model name is just the file name. It is extracted as a separate component since it often embeds meaningful domain hints (e.g., terms such as *isolette*, *drone*, or *car*), which provide useful semantic information for clustering and help distinguish models that may exhibit similar structures but belong to different application domains. The *text-based* representation of an *aadl* model consists of a text with the concatenation of all its node names.

SSC does not include other names (e.g., attribute names) because attribute names are often generic (for instance, they use labels like name, from, to, etc.), carrying little information about the purpose of the model.

3.2. Calculating Similarity Matrix

The comparison of two architectural models is obtained by combining three similarity indices: the *structural similarity*, the *semantic similarity of model names*, and the *semantic similarity*

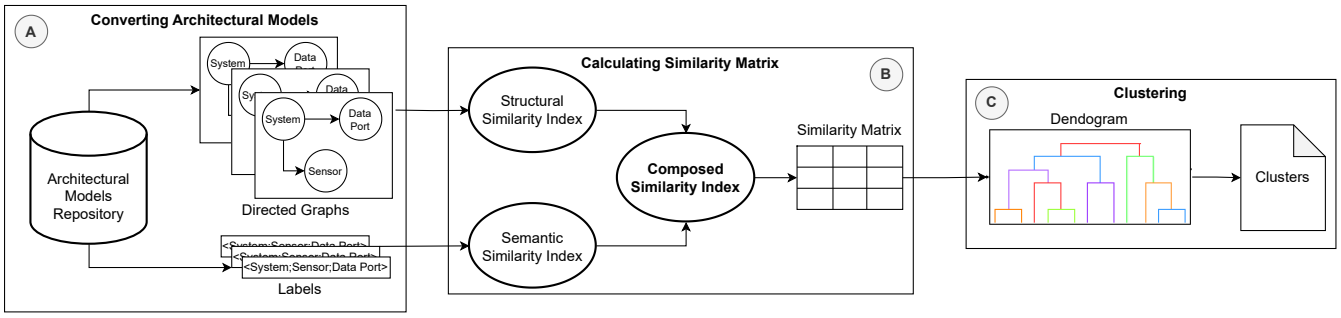

Figure 2 Structural-Semantic Clustering.

of model content. This combination allows for a comprehensive evaluation of the similarity between models, from both their internal structure and their application context perspectives.

The computation of the *structural similarity* among two models $aadl_1$ and $aadl_2$ is based on the comparison of their graph-based representations G_1 and G_2. Intuitively, the greater the overlap between these two graphs, the more similar the models' structures are. To implement this intuition, SSC exploits the notion of *Maximum Common Subgraph (MCS)* (Duesbury et al. 2017) between the compared models. The MCS represents the largest shared structure between two graphs. We adopt an MCS-based similarity because it directly captures the largest shared structural patterns between models, aligning with our goal of identifying reusable architectural fragments. Alternatives such as graph edit distance (Gao et al. 2010) or kernel- and embedding-based methods (Cai et al. 2018) trade structural interpretability for scalability, whereas MCS provides a clearer and more faithful representation of structural similarity.

In software or system architectures, solutions may share a similar backbone with differences in some components due to specific requirements. Measuring similarity based on the common structure allows capturing this continuity across domains, distinguishing the common and the specific parts of a model.

It is worth noting that common nodes in the maximum subgraph represent the presence of the same type of components in both models, playing the same role and interacting with other components, again playing consistent roles. Intuitively, this captures the architectural patterns that may repeat across models.

SSC can exploit the notion of MCS in three different ways for the computation of the structural similarity. These alternatives consider the weight of the shared part with respect to the overall graph sizes differently. We compare these strategies empirically in the next section.

Let $G_1 = (V_1, E_1)$ and $G_2 = (V_2, E_2)$ be two graphs, and let $H = \text{MCS}(G_1, G_2) = (V_H, E_H)$ denote their maximum common subgraph. Structural similarity is derived by combining a node-based and an edge-based similarity component computed from H. The three structural similarity measures considered in this work are defined as follows:

- $StructSim_{max}(G_1, G_2) = \frac{1}{2}\left(\frac{|V_H|}{max(|V_1|,|V_2|)} + \frac{|E_H|}{max(|E_1|,|E_2|)}\right)$. This formula gives higher weight to differences between the graphs by normalizing the size of the MCS with respect to the largest graph among the ones that are compared. As a result, the similarity will be lower if there is a significant disparity between the sizes of the two graphs.
- $StructSim_{min}(G_1, G_2) = \frac{1}{2}\left(\frac{|V_H|}{min(|V_1|,|V_2|)} + \frac{|E_H|}{min(|E_1|,|E_2|)}\right)$. This formula gives higher weight to similarities, as it normalizes with respect to the smallest graph. It is particularly useful when the goal is to focus on graph inclusion.
- $StructSim_{avg}(G_1, G_2) = 0.5 \times StructSim_{max}(G_1, G_2) + 0.5 \times StructSim_{min}(G_1, G_2)$. This formula balances the effect of graph sizes, combining the two indices defined above.

The computation of the *semantic similarity* among two models $aadl_1$ and $aadl_2$ is based on the comparison of their names, $name_1$ and $name_2$, and the text-based representations, txt_1 and txt_2. Both of them are compared as follows:

1. *Word list cleaning*: The words in each textual representation are preprocessed to remove special characters, numbers, and generic keywords (e.g., system, instance, device) that do not significantly contribute to expressing the components' semantics so that only the likely relevant terms are retained for comparison.
2. *Extraction of the semantic vectors*: The cleaned terms are mapped into semantic vectors that numerically represent their meaning in a multidimensional space. This can be done either with traditional statistical approaches such as TF-IDF, or with modern embedding methods such as FastText (Athiwaratkun et al. 2018), RoBERTa (Y. Liu et al. 2019), MiniLM (Enevoldsen et al. 2025), and MPNet (Enevoldsen et al. 2025), which can capture deeper contextual semantics. We empirically compare these embeddings on our evaluation.
3. *Calculation of vector similarity*: Once the semantic vectors are obtained, the similarity between pairs of architectures is computed using cosine similarity, resulting in a semantic similarity score that reflects their closeness in meaning.

The *composed similarity index* is obtained as a weighted combination of the indices about the structure of the models, the semantics of the models' names, and the semantics of the models' content:

$$\textit{composed similarity} = w_{\text{str}} \times \textit{structural similarity} + w_{\text{name}} \times \textit{model-name similarity} + w_{\text{text}} \times \textit{text-based similarity}$$

where w_{str} is the weight assigned with the structural similarity index, w_{name} and w_{text} are the weights assigned with the semantic similarity index, with $w_{\text{str}} + w_{\text{name}} + w_{\text{text}} = 1$. We systematically investigate multiple combinations of these weights in

our evaluation. The similarity values between the model pairs are organized into a similarity matrix, where each element s_{ij} represents the similarity between model i and model j.

3.3. Clustering

To produce the final set of clusters considering possibly different aggregation criteria, SSC uses hierarchical agglomerative clustering, which consists of an unsupervised clustering technique that builds a hierarchy of clusters through an iterative process of merging the most similar clusters.This approach is particularly useful as it allows identifying clusters of different sizes, offering the ability to extract information at various levels of granularity. For instance, it can produce a few broader clusters that group models with general similarities or a greater number of more specific clusters, focused on particular characteristics of the models. This flexibility allows for adapting the analysis to specific goals. This is particularly important in our domain, since analysts may have different needs depending on the case: sometime they may need to identify a large set of loosely coupled models relative to a same subject area, to explore how systems in a certain context are architected, another time they may need to identify a small set of models all solving the same problem to borrow ideas for the design of a new system.

The starting point of hierarchical clustering is the similarity matrix, where each element s_{ij} represents the similarity between model i and model j. To apply the clustering algorithm, the similarity matrix is converted into a distance matrix D according to the formula $d_{ij} = 1 - s_{ij}$. Thus, a high similarity corresponds to a low distance and vice versa. The resulting distance matrix is symmetric and has zeros on the diagonal (the distance of a model with itself is 0).

Hierarchical agglomerative clustering begins by considering each model as a separate cluster. At each iteration, the two most similar clusters are merged into a single cluster. This process continues until all models have been grouped into a single cluster. The distance between clusters during the merging process is measured with the Ward's linkage method, which uses the Euclidean distance between the centroids, weighted by the size of the cluster, as a distance metric. This method minimizes the total variance within the combined clusters.

The result of hierarchical agglomerative clustering can be visualized with a *dendrogram*, a tree diagram where leaves represent individual models and internal nodes represent the cluster merge points (see Figure 4).

The benefits of using a dendrogram to visualize the hierarchical structure is two-fold. On the one hand, it supports determining the optimal number of clusters. By cutting the dendrogram at a certain height (distance), it is possible to partition the data into a specific number of clusters. Analyzing the distances between merges helps identify the most appropriate cutting point. On the other hand, it helps interpret relationships between models. Specifically, it provides an overview of the similarities between models, highlighting which models or groups of models are more similar to each other. Once a user has determined the wanted cut level, the corresponding clusters can be reported in the output.

4. Dataset Construction

4.1. Data Collection and Filtering

To systematically investigate the effectiveness of SSC, we created a large dataset of architectural models collected from the wild. In particular, we mined GitHub repositories to identify publicly available AADL models.

We implemented a tool that leverages GitHub's public APIs to automatically identify repositories containing architectural models stored in `aadl`, `aadl2` and `aaxl2` formats. The tool uses this list to automatically clone the identified repositories and extract the architectural models, creating an initial repository of 1,278 models for our study.

We validated the models using the validator in the OSATE plugin[2], removing models with syntactic errors, unresolved dependencies, or inconsistencies with respect to the AADL metamodel. We further inspected the valid models, eliminating toy examples such as models with a single top-level component and no subcomponents, connections, or property specifications. We ended up establishing a dataset of 1,202 validated architectural models mined from GitHub, with an average of 29.4 nodes and 34 edges. As shown in Figure 3, the distributions of both nodes and edges are highly right-skewed, with the majority of models containing fewer than 50 components and connections. At the same time, the dataset exhibits a long-tailed behavior, with a small number of significantly larger models. In particular, the largest model contains 934 nodes and 1,188 edges, highlighting the presence of extreme cases. The similar shapes of the two distributions suggest that the number of connections scales proportionally with the number of components. These models come from various sectors, including aerospace, healthcare, automotive, and other relevant fields, ensuring diversity.

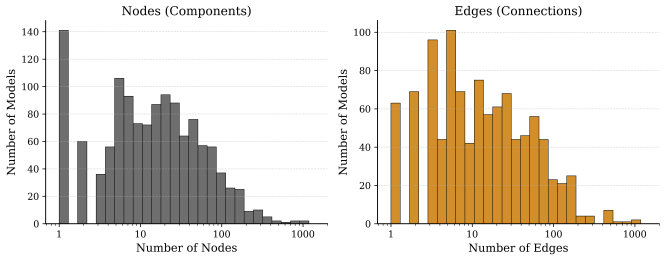

Figure 3 The distribution of model sizes.

Figure 4 shows an example dendrogram produced by SSC with the models in the dataset. The analysis can use the dendrogram by selecting a cut threshold, obtaining either high-level domain groupings, if the cut is high, or fine-grained clusters of architectures addressing similar goals, if the cut is low.

4.2. Ground Truth Creation

We constructed the ground truth according to two criteria reflecting common analysis scenarios: clustering models by *application domain*, to support the exploration of architectures within a broad sector, and clustering models by *specific problem*,

[2] https://osate.org/

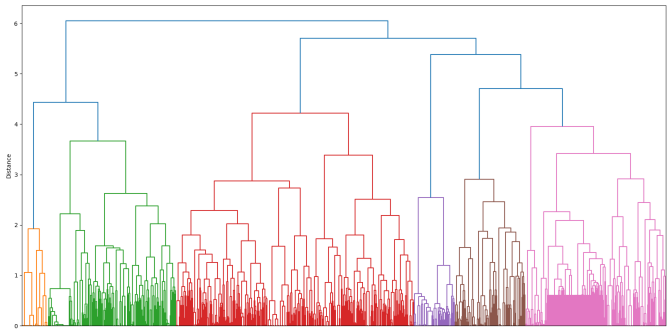

Figure 4 Dataset classified with SSC.

to identify architectures addressing the same concrete design problem and enable targeted reuse.

Two authors have been independently involved as reviewers in the inspection and classification process. In each round, the models were reviewed by both authors, who examined the files independently and then met to discuss their classifications. Any ambiguity or discrepancy was addressed collaboratively during these follow-up sessions, where the authors discussed their reasoning and made adjustments to ensure consistency. Over time, the reviewers reached a consensus, and all agreed on the resulting ground truth.

The set of *application domains* used to classify the models emerged incrementally during the analysis and was finalized in the consensus phase. We identified a total of 13 frequent domains, such as aerospatial, automotive, smart building, etc. During the annotation process, some models were initially associated with multiple candidate domains. These cases were carefully reviewed and resolved through discussion and consensus among the annotators, resulting in a single, most appropriate domain assignment for each model. The resulting classification has been independently validated by two additional authors, who reviewed ambiguous cases and participated in a follow-up discussion to finalize the ground truth dataset.

To obtain the ground truth that groups models according to their *specific problem*, we considered the objective and functionalities present in each architecture to create the groups. In this context, a *specific problem* denotes the concrete system-level functionality or design problem addressed by an architecture (i.e., *what the system is intended to achieve*), independently of the application domain (i.e., *where it is deployed*). For example, in the automotive domain, all models related to cruise control were grouped together; in the aviation domain, models related to temperature management in avionics systems were clustered. This approach allowed for the creation of more targeted and homogeneous clusters, representing the specific functionalities of the architectures with greater precision. We adopted the same process used for the application domain ground truth, finally classifying models in 49 clusters.

Throughout the process, disagreements were resolved through discussion and clarification, ensuring consistency across the classifications. To quantify the level of agreement between the annotators before reconciliation, we computed *Cohen's Kappa* coefficient (Artstein 2017), a standard metric for inter-annotator reliability. The resulting κ score was 0.764 for the application domain and 0.893 for the specific problem, indicating strong agreement between the two reviewers.

We ultimately released the largest curated ground truth, 1,202 AADL models, clustered according to two criteria.

5. Evaluation

We assess the proposed method by empirically investigating alternative strategies to compute structure, semantic, and hybrid similarities.

RQ1 (Embedding). How is the quality of the clusters influenced by the embedding method used to compute the semantic similarity?
This research question aims to compare different embedding methods for calculating semantic similarity to identify the one that provides the best performance and will be used in subsequent research questions.

RQ2 (Structural). How is the quality of the clusters influenced by the method used to calculate the structural index?
This research question aims to compare the effectiveness of the structural similarity indices (avg, min, max) to determine which method yields better results, and this will be used in subsequent research questions.

RQ3 (Hybrid). How is the quality of the clusters influenced by the combination of structural and semantic weights?
This research question explores the sensitivity of clustering with respect to the choice of the weights assigned to the structural and semantic similarity indices.

RQ4 (Cut-off). What impact does the choice of the number of clusters and cut-off criteria have on the clustering results?
Finally, this research question investigates how the choice of the number of clusters affects the clustering results.

5.1. Evaluation Metrics

To evaluate the quality of the clusters, we use the Adjusted Rand Index (ARI), a widely used index to measure the quality of a cluster with respect to a ground truth (Steinley 2004). The ARI is derived from the Rand Index (RI) (Warrens & van der Hoef 2020), which quantifies the similarity between two partitions of a dataset, such as a clustering result and the ground truth. The formula for the RI is: $RI = \frac{a+b}{\binom{n}{2}}$, where a is the number of pairs of elements assigned to the same cluster in both partitions, b the number assigned to different clusters in both, and n the total number of elements. ARI normalizes the RI by the chance of agreement expected under random cluster assignment: $ARI = \frac{RI - E[RI]}{max(RI) - E[RI]}$, where $E[RI]$ is the expected value of the Rand Index for random assignments. The ARI ranges between $[-1, 1]$, where 1 indicates perfect agreement with the ground truth, 0 indicates agreement equivalent to random assignment, and a negative value indicates a result worse than chance.

We use the two ground truths we designed to measure the quality of the clustering with respect to different aggregation objectives, one more generic (clustering for application domains) and one more specific (clustering for specific problems). We do

not use any internal clustering metrics (e.g., the silhouette) that simply measure the level of separation of the clusters because they could be easily optimized artificially without reflecting the intended objective of the clustering. On the contrary, the goal of the clustering here is entirely related to the need to support multiple aggregation levels according to the goal of the analysts, which could be measured only by referring to a ground truth.

In addition to quantitative metrics, we conduct a *qualitative evaluation* of the clusters reconstructed by the best performing configurations of SSC. We use manual analysis to identify lessons learned about what the models can effectively detect and what may escape analysis, offering a more in-depth understanding of the clustering results and the underlying dynamics.

5.2. Experimental Procedure

To answer RQ1, we compare five embedding methods used to compute semantic similarity. We consider methods of different kinds: *TF-IDF* (Salton & Buckley 1988), a classical statistical approach based on word frequency, *FastText* (Athiwaratkun et al. 2018) (`crawl-300d-2M`), which extends this idea by incorporating subword information to better capture the semantics of rare or unseen words, *RoBERTa* (Y. Liu et al. 2019) (`roberta-base-nli-stsb`), which is a transformer-based model that improves over BERT with more robust training, *MiniLM* (Enevoldsen et al. 2025) (`all-MiniLM-L6-v2`), which provides a lighter and faster alternative while retaining good accuracy, and *MPNet* (Enevoldsen et al. 2025) (`all-mpnet-base-v2`), which combines masked and permuted language modeling to enhance contextual understanding. These methods are systematically evaluated to identify the one that offers the best performance, which will then be employed in the subsequent research questions.

To answer RQ2, we systematically assess the quality of the clusters returned by $StructSim_{max}$, $StructSim_{min}$, and $StructSim_{avg}$, for every possible value of the weights (considering every combination from 0 to 1 with step 0.05), and for every possible cut threshold (considering every combination from 0 to 1 with step 0.05). We consider both the application domain and the specific goal ground truths. The best configuration is used to address the subsequent research questions.

To answer RQ3, we select from RQ1 and RQ2 the configurations that work best for the two considered ground truths, and we systematically investigate the impact of the weights. In particular, we measure the quality of the clusters for every possible combination of $(w_{str}, w_{name}, w_{text})$, with each weight ranging from 0 to 1 in increments of 0.05, under the constraint that $w_{str} + w_{name} + w_{text} = 1$.

Finally, *to answer RQ4*, we use the best combination in terms of embedding method, structural comparison strategy, and relative weights of the components, and we systematically measure the quality of the clusters for changing values of the cut, ranging from 0 to 1 with a step of 0.05. This analysis reveals findings on the stability of the clusters and the optimal set of clusters that can be generated.

6. Results

6.1. RQ1 - Embedding

To address this research question, we compared five embedding methods (TF-IDF, FastText, RoBERTa, MiniLM, and MPNet) by evaluating the clustering results against two different ground truths. The comparison highlights both the central tendency (median ARI) and the variability (spread) of the results.

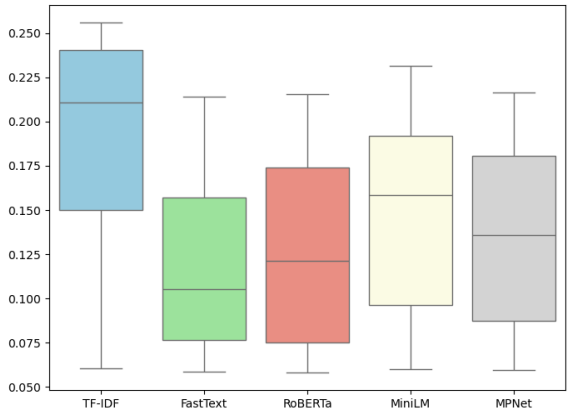

Figure 5 *ARI* values for different embedding methods (application domain ground truth).

Regarding the application domain ground truth, as shown in Figure 5, TF-IDF achieves the highest median ARI among all methods. Meanwhile, more modern methods achieve lower median ARI scores, typically ranging between 0.10 and 0.18. This suggests that TF-IDF provides better central performance and yields more reliable clustering quality in the generic domain.

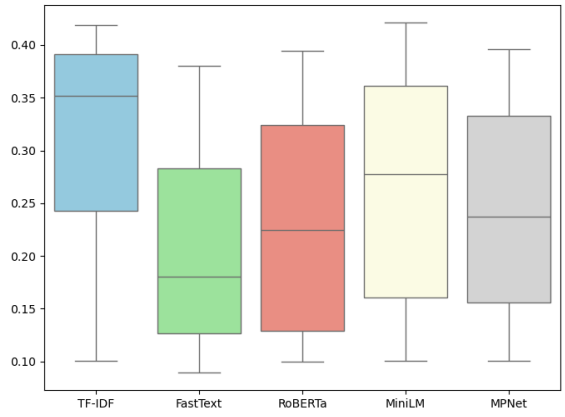

Figure 6 *ARI* values for different embedding methods (specific problem ground truth).

The results for the specific problem ground truth, shown in Figure 6, further confirm the superiority of TF-IDF. In this setting, TF-IDF again achieves the highest median ARI, close

to 0.35, while the transformer-based methods perform moderately well with median values around 0.20–0.30. MiniLM is the second-best option, but it still performs worse than TF-IDF. FastText performs the weakest among the tested methods. Overall, TF-IDF demonstrates both higher effectiveness and greater robustness across runs.

Although modern transformer-based embedding methods (RoBERTa, MiniLM, and MPNet) offer competitive alternatives, they fall short of TF-IDF. Therefore, in the following research questions, we adopt TF-IDF as the embedding method for computing semantic similarity.

Answer to RQ1

TF-IDF achieves the best ARI scores in both the generic and the specific problem ground truths.

6.2. RQ2 - Structural

Figures 7 and 8 show ARI values for each structural index: $StructSim_{avg}$, $StructSim_{min}$, and $StructSim_{max}$.

Regarding the application domain ground truth (Figure 7), we observe that $StructSim_{max}$ achieves the highest median ARI, slightly higher than $StructSim_{avg}$ and clearly higher than $StructSim_{min}$. This indicates that normalizing by the largest graph leads to more discriminative similarity values, which in turn produce more accurate clusters. $StructSim_{avg}$ follows closely, suggesting that averaging the two indices is still able to capture relevant structural signals. By contrast, $StructSim_{min}$ underperforms, since its normalization with respect to the smallest graph tends to overestimate similarities and thus reduces the ability to separate different application domains.

We performed statistical significance tests on the ARI distributions. The difference between $StructSim_{max}$ and $StructSim_{avg}$ is statistically significant according to the Wilcoxon signed-rank test (Wilcoxon 1992) with $p = 5.37 \times 10^{-3}$, but the effect size is small (Cohen's $d = 0.21$) (Cohen 2013). This suggests that, although $StructSim_{max}$ consistently outperforms $StructSim_{avg}$, the practical improvement remains limited. In contrast, both $StructSim_{max}$ and $StructSim_{avg}$ significantly outperform $StructSim_{min}$ ($p < 10^{-6}$), with moderate effect sizes (Cohen's $d \approx 0.4$), indicating that normalization based on the smallest graph consistently degrades clustering performance.

For the specific-problem ground truth (Figure 8), the superiority of $StructSim_{max}$ is also evident. The reason is that by penalizing size differences more heavily, $StructSim_{max}$ avoids artificially inflated similarities that would otherwise occur with $StructSim_{min}$ in cases of partial graph inclusion. As a result, it is better aligned with the stricter notion of similarity required when grouping models that solve the same specific problem. $StructSim_{avg}$ again provides intermediate results, confirming its role as a compromise between the two extremes.

These observations are further supported by statistical analysis. The improvement of $StructSim_{max}$ over $StructSim_{avg}$ is statistically significant (Wilcoxon signed-rank test, $p < 10^{-15}$), with a moderate effect size (Cohen's $d = 0.63$), indicating a consistent and practically meaningful gain. Moreover,

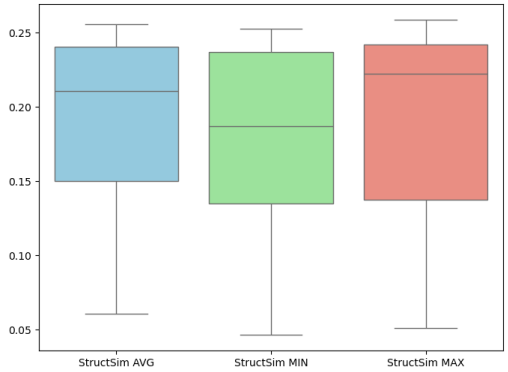

Figure 7 Distribution of ARI values for different structural similarity formulas and the application domain ground truth.

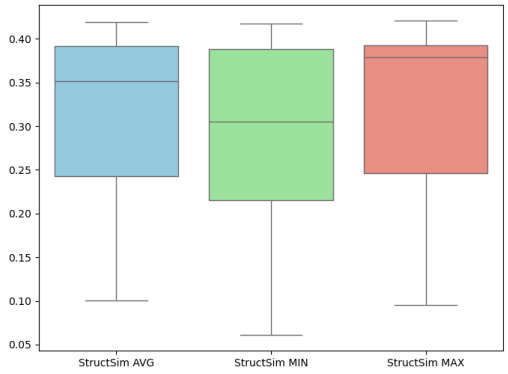

Figure 8 Distribution ARI values for different structural similarity formulas and the specific problem ground truth.

both $StructSim_{max}$ and $StructSim_{avg}$ significantly outperform $StructSim_{min}$ ($p < 10^{-24}$), with large effect sizes (Cohen's $d \approx 0.87$). Notably, these effect sizes are substantially larger than those observed for the application domain ground truth, suggesting that the choice of normalization becomes increasingly critical when the notion of similarity is more fine-grained.

$StructSim_{max}$ penalizes model size imbalance by normalizing with respect to the larger graph. This leads to lower similarity when there is a significant disparity between graph sizes, preventing small graphs from appearing overly similar to larger ones due to partial inclusion. As a result, $StructSim_{max}$ provides more conservative similarity values, which explains its better clustering performance.

Since the ARI values with the specific-problem ground truth are higher than the values with the application domain ground truth, we can observe that SSC is more effective in grouping architectures that solve closely related tasks than in clustering models according to their broader application domain.

Answer to RQ2

The results show that $StructSim_{max}$ consistently outperforms the other alternatives across both ground truths.

6.3. RQ3 Hybrid

This research question investigates the actual contributions of the structural, model name, and text-based components to establish if there is any gain in combining these three components. Note that we use TF-IDF, since it performed best in RQ1, and $StructSim_{max}$, since it performed the best in RQ2. Hence, for every pair $(w_{\text{str}}, w_{\text{name}})$ in the heatmaps, w_{text} is implicitly set to $1 - w_{\text{str}} - w_{\text{name}}$ (feasible region shown as the triangle). We report ARI for two ground truths: *application domain* and *specific problem* (Figure 9 and Figure 10).

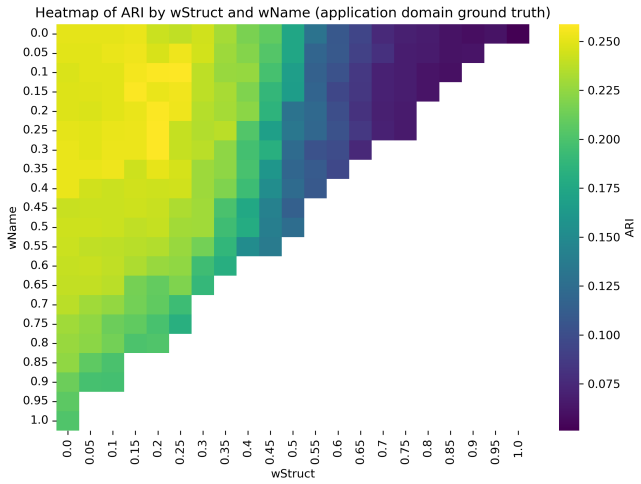

Figure 9 Heatmap of ARI over $(w_{\text{str}}, w_{\text{name}})$ for the *application domain* ground truth.

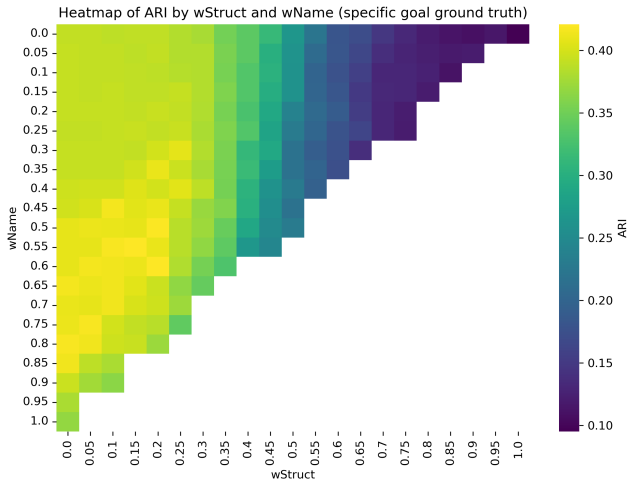

Figure 10 Heatmap of ARI over $(w_{\text{str}}, w_{\text{name}})$ for the *specific problem* ground truth.

For the application domain ground truth (Figure 9), the best ARI is 0.2588 at $(w_{\text{str}}, w_{\text{name}}) = (0.25, 0.10)$, which implies a large residual weight for semantics $w_{\text{text}} = 0.65$. The warm region concentrates where w_{name} is small (≤ 0.2) and w_{str} is moderate (≈ 0.2–0.3). For the specific problem ground truth, the best ARI is 0.4209 at $(w_{\text{str}}, w_{\text{name}}) = (0.15, 0.55)$ with $w_{\text{text}} = 0.30$ (Figure 10). High ARI bands appear when w_{name} is larger (≈ 0.45–0.6) and w_{str} stays low (≈ 0.1–0.2).

For *application domain* clustering, the text has the highest weight in the optimal configuration, suggesting that domain information is primarily encoded in textual cues present in component descriptions and identifiers, while model names vary widely and are weakly aligned with broad domains. For the *specific problem* clustering, a high w_{name} markedly improves ARI, indicating that model names tend to carry information that is highly discriminative at this granularity.

In both heatmaps, ARI degrades when w_{str} becomes too large. Over-emphasizing structure reduces the capacity to exploit the more predictive signals (i.e., texts and names). Practically, structure acts as a stabilizer: moderate w_{str} (≈ 0.15–0.30) helps, but higher values sacrifices either w_{text} or w_{name} and hurts ARI.

The maximum ARI for *specific problem* (0.4209) is substantially higher than for *application domain* (0.2588). This gap implies goals are more cleanly separable by our signals, whereas domains are fuzzier and require stronger semantic context to overcome noisy or inconsistent naming (Figures 9, 10).

Answer to RQ3

To cluster according application domains, it is better to prioritize semantics (w_{text} large), keeping w_{str} modest, and w_{name} small. We observed the optimum at $(w_{\text{str}} = 0.25, w_{\text{name}} = 0.10, w_{\text{text}} = 0.65)$. When targeting specific problems, it is better to allocate a larger portion to names. We observed the optimum at $(w_{\text{str}} = 0.15, w_{\text{name}} = 0.55, w_{\text{text}} = 0.30)$.

6.4. RQ4 Cut-Off

RQ4 aims to evaluate the impact of the number of clusters and the cutting criteria on the clustering results for both ground truths. In particular, we considered every possible aggregation of clusters that can be returned by SSC for different cut values. We study the performance of $StructSim_{max}$ with the weight combination $(w_{\text{str}} = 0.25, w_{\text{name}} = 0.10, w_{\text{text}} = 0.65)$ for the application domain ground truth and $(w_{\text{str}} = 0.15, w_{\text{name}} = 0.55, w_{\text{text}} = 0.30)$ for the specific problem ground truth, since these configurations performed the best.

Results for the application domain ground truth are shown in Figure 11. The ARI reaches its highest value of 0.2588 at 33 clusters. In the range between roughly 30 and 110 clusters, performance remains relatively stable, indicating that moderate partitioning still preserves clustering quality. However, beyond this point, ARI consistently decreases as the number of clusters increases, highlighting that excessive fragmentation splits well-represented domains into smaller subdomains, thereby degrading the quality of clustering.

Figure 12 shows how ARI values change with the number

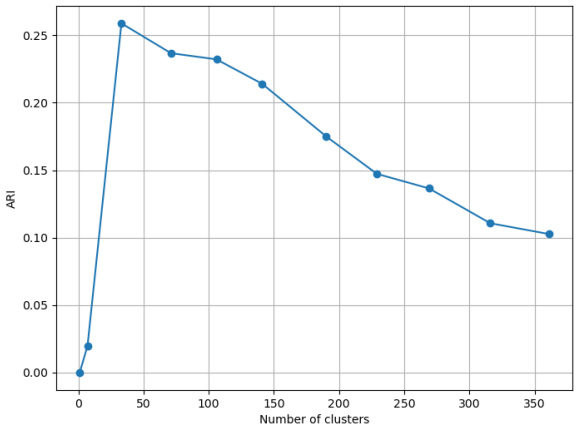

Figure 11 Trend of ARI values for a changing number of clusters in the case of the application domain ground truth.

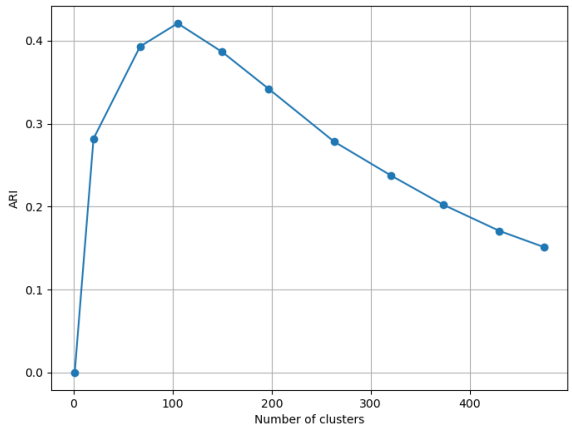

Figure 12 Trend of ARI_{GT} w.r.t. the cut with specific problem *Ground Truth*.

of clusters for the specific problem ground truth. Initially, an increasing number of clusters improves the quality of results, achieving its peak at 0.4209 with 105 clusters. This is a substantially higher maximum compared to the application domain case. The peak of high ARI values is longer: a range between about 65 and 150 clusters all yield strong performance. After this interval, ARI gradually decreases as fragmentation increases.

Results show again a better capability of SSC to aggregate models addressing a specific problem, being less sensitive to the choice of the number of clusters and obtaining better ARI overall. This might be because the defined methodology is more effective at recognizing specific patterns and terms within the models, rather than discovering a more general categorization.

Answer to RQ4

SSC demonstrated a better capability in discovering specific problem-oriented clusters than general domain-oriented clusters, where a smaller sensitivity to the choice of the cut-off value is also observed.

6.5. Qualitative Analysis

This section provides a qualitative discussion of the clustering configurations that achieved the most significant results, with the weight combination for the application domain ground truth and for the specific problem ground truth.

Application Domain Ground Truth. Under the application domain labeling, SSC generated 33 clusters, indicating that grouping architectures according to broad, abstract criteria remains challenging. Some domains are easier to recognize, for instance, models related to temperature management form coherent groups with a good balance between structural and functional similarity. However, several hard cases highlight both cross-domain confusion and insufficient abstraction in the current representation.

False positives often arise when systems across different domains exhibit high similarity scores. For example, models from the `aadl-qa_test_package_refined_Test_System_implX` group achieved similarity values above 0.95 but were assigned to distinct domains such as `AUTOMOTIVE` and `SMART_BUILDING`. A similar issue occurs in the `Sunseeker` case, where two implementations (`polyorb-hi-ada` vs. `polyorb-hi-c`) reached similarity values above 0.92, but were clustered under `SMART_BUILDING` and `EXAMPLE`. Likewise, comparisons between `Isolette` and `Nursery Room` models (`SMART_BUILDING` vs. `SMART_HEALTH`, similarity $\approx$ 0.87) illustrate the difficulty of distinguishing closely related domains that address similar environmental and safety concerns. These cases suggest that the current feature space is insufficiently discriminative to separate structurally similar architectures that belong to different semantic fields, suggesting that additional contextual information may be required.

False negatives, by contrast, occur when models within the same domain show extremely low similarity scores (often below 0.003). For instance, `Tank-Pressurized` was assigned very low similarity with both `aadl-qa_issue182_testX` and `CarSystem`, despite all belonging to `AUTOMOTIVE`. This highlights that the representation fails to capture underlying logical commonalities, likely because it is overly sensitive to superficial variations such as naming conventions, hierarchical depth, or architectural complexity.

Identifying Models of the Same Application Domain

While aggregating models by application domain may appear intuitive, manual inspection confirms that it remains difficult in practice. SSC discovers some meaningful groups, but also misses desirable relations among diverse architectures within the same domain, suggesting that more abstract conceptual modeling is needed.

Specific Problem Ground Truth. Under the specific problem labeling, clustering is generally more coherent, as models often share both architectural patterns and functional objectives. For instance, neonatal incubator monitoring systems show strong structural alignment reflecting critical sensor management, while cruise control models form consistent groups around vehicle control architectures. Semantic aspects also help cluster models related to drones, resource management, and cybersecurity, where modularity and security constraints drive architectural similarity.

False positives mainly occur between closely related subdomains, reflecting cases of *over-segmentation* where manual labels are more fine-grained than what structural similarity can reasonably distinguish. For example, `Isolette dual sensor` and `Isolette single sensor` models (similarity ≈ 0.90) were separated into `THERMOSTAT_DUAL_SENSOR` and `THERMOSTAT_SINGLE_SENSOR`, even though their architectures differ only marginally. Similarly, in the `Sunseeker` case, models reached similarity values near 0.87 but were divided between `SUNSEEKER_CONTROL_SYSTEM` and `SCENARIO_PRODUCER`, reflecting different roles within the same system.

False negatives further reinforce the limitations observed in the application domain setting. Numerous pairs within the same cluster obtained similarity scores close to zero, for instance `self-driving car integration` vs. `remoteVehicleOBE` ($\approx$ 0.0039), or `ocarina_t11_system` vs. `example_cca_cma1` ($\approx$ 0.004), despite belonging to the same domain. This demonstrates that the current approach fails to capture functional equivalence when structural realizations diverge strongly, an issue compounded by the fine granularity of specific goal labels. Improving the representation to recognize equivalence at the functional level will be essential, potentially by incorporating domain knowledge such as ontologies or knowledge graphs.

Identifying Models Addressing the Same Specific Problem

SSC is effective in clustering architectural models according to the addressed problem, producing useful aggregations that analysts can exploit, although heterogeneous realizations of the same function remain challenging.

6.6. Operational Feasibility

In addition to effectiveness, we evaluate the computational cost and scalability of the proposed similarity computation.

The framework was executed on a standard CPU environment (2 cores) using a dataset of 1,202 AADL models, resulting in 723,003 pairwise comparisons. On average, each pairwise comparison required 17.1 ms, with a median of 10 ms and a maximum of 52.6 seconds. The system achieved a throughput of 111.8 comparisons per second, demonstrating efficient parallel execution. Regarding memory usage, the peak memory consumption during similarity computation remained below 2 GB. This indicates that the approach is feasible and scalable on commodity hardware without requiring specialized resources.

6.7. Threats to Validity

The main internal threats to validity concern the creation of the dataset, the creation of the ground truth, and the computation of the ARI metric. The creation of the dataset for the study is based on the mining of an extensive collection of models from GitHub repositories. To guarantee the quality of the resulting models, we manually inspected them to discard incomplete and toy models, in addition to discarding the ones with syntactic problems and unresolved dependencies. Overall, we expect this set of activities to ensure a high-quality dataset, publicly available for third-party inspection and use (Tran et al. 2025).

The creation of the two ground truths is the result of a significant effort by the authors of this paper. Although there might be some subjectivity in the classification of specific models, the agreement among the inspectors provides a good level of confidence in the quality of the result.

The main external threats to validity concern the possibility of generalizing our findings to additional architectural model files and additional clustering objectives. In our study, we considered AADL files. Although we had to choose a specific target language, we do not see any reason why our findings should be limited to AADL files, and could not be extended to architectures represented with other architectural languages.

6.8. Insights for Practitioners and Researchers

Aggregating architectural models by application domain can be challenging. Our results show that creating large groups of architectural models all referring to the same general application domain can be difficult. Some models might be misclassified, and the resulting clusters would be imperfect, requiring manual revision by experts before they can be used. *More work is needed to design clustering algorithms that can be equally effective at multiple levels of abstractions*.

Discovering clusters of architectures that solve the same concrete problem is feasible. Our results show that architectures that address the same concrete problems could be feasibly discovered and aggregated. This is particularly valuable for *practitioners and analysts that can exploit this capability to solve new problems by reusing the knowledge that derives from the analysis of existing architectures*.

Simple semantic baselines remain highly competitive. Interestingly, our experiments show that a lightweight TF-IDF representation often outperforms more complex embedding-based models in clustering architectural artifacts. This suggests that architectural model repositories still show sparse and keyword-driven semantics, and that strong results can be achieved with interpretable and efficient text-based techniques. *Future work is*

needed to design embeddings better aligned with the vocabulary and structure of architecture modeling languages.

Meaningful clustering requires combining structural and semantic information. Our results show that structural and semantic information alone provide only a partial view: architectures solving different problems may share similar structures, while architectures solving the same problem may differ structurally. This issue is further amplified when mixing system-level and component-level models. *A combined perspective on structure, semantics, and abstraction level is necessary to produce useful clustering results*.

Functionally-equivalent but structurally-different models are hard to discover. Architecturally diverging solutions of the same problem are hard to detect, since the different structures of the models complicate functional analysis. To better handle these cases, additional sources of information should be considered. For instance, the analysis could be informed by the analysis of the source code or of any documentation available, to reduce the possibility that diverging structures become a confounding factor.

7. Related Work

Various methods for model comparison have been developed, including classification and clustering. Early studies focused on classification using *labeled data*, training neural networks or machine learning classifiers to identify predefined categories. Using a labeled dataset of 555 metamodels (Babur 2019), AURORA (Nguyen et al. 2019) employed a feed-forward neural network, while memoCNN (Nguyen et al. 2021) used a convolutional neural network. Lopez et al. proposed MAR, a structure-based search engine for model comparison (López & Cuadrado 2020). Khalilipour et al. applied various machine learning classifiers for model categorization (Khalilipour et al. 2022), and López et al. systematically compared encoding techniques for model classification (López et al. 2022). Graph-based neural models were also built to encode abstract syntax trees, control flow, and data flow into a unified embedding for functional similarity detection (J. Liu et al. 2023). Differently from SSC, these techniques generally depend on labeled datasets, which are not always available across diverse architectural domains.

Clustering techniques group unlabeled models based on similarities *without predefined labels*. Several studies analyzed *structural relationships* for clustering. Graph kernels (Clarisó & Cabot 2018) and n-grams (Babur & Cleophas 2017) have been used in metamodel comparison and hierarchical clustering of Ecore models. Aroma clusters method bodies to recommend relevant code snippets (Luan et al. 2019), but its strict definition of similarity may struggle with broader architectural similarities. More recently, SARIF (Zhang et al. 2023) addressed architecture recovery from codebases by combining dependency graphs, textual information, and folder structure, highlighting the value of integrating multiple sources of architectural evidence.

Clustering approaches focusing on *semantic information* often employ NLP techniques to capture linguistic relationships within model descriptions. Some methods represent descriptions as high-dimensional vectors (Babur et al. 2016). López et al. (López et al. 2023) trained word embeddings for the MDE domain, called WordE4MDE, while an Apache Lucene-based system was introduced to index and search metamodels (Rubei et al. 2021). Structural or semantic information alone is insufficient to capture clusters of related models, so the interplay of the two factors must be considered.

The *integration of both semantic and structural information* has also been explored. Structural and semantic similarity functions have been combined with specific thresholds to cluster metamodels (Basciani et al. 2015, 2016), and community detection has been applied to cluster software components (Sas & Capiluppi 2021), although the latter used semantic features only for validation. For classification, Khalilipour et al. introduced a hybrid approach (Khalilipour et al. 2024) combining textual and structural information, and LLM-based prompting (Rukmono et al. 2024) was used to classify Java methods into components of a reference architecture.

Unlike these works, which mainly extract architecture from source code and may be constrained by syntax-based representations, SSC directly analyzes high-level architectural models shared on GitHub. This enables a more flexible integration of semantic and structural information, generating aggregations that architects can exploit to design new systems.

8. Conclusions

Open-source repositories are valuable for architects seeking to reuse knowledge from design models, yet few studies have effectively extracted and organized this information. To address this gap, we proposed a structural-semantic clustering applied to 1,202 architectural models extracted from GitHub to distill useful aggregations. The core idea of combining structural and semantic information can be extended with information derived from code, comments, or documentation. Future work concerns analyzing the extracted clusters to generate artifacts for various purposes (e.g., distilling reference architectures), extending the tool to other architectural languages (e.g., ArchiMate), and exploring additional clustering strategies on the computed similarity matrices (e.g., spectral clustering or k-medoids).

References

Akthar, S. R., Islam, M. R., Hasan, M. B., Siddiqua, M. S., Haque, S. I., Saad, J. A., … Hasan, M. (2024). Overcoming obstacles in model-driven engineering: Lessons from the software industry. In *International conference on software technologies.*

Artstein, R. (2017). Inter-annotator agreement. In *Handbook of linguistic annotation.*

Athiwaratkun, B., Wilson, A. G., & Anandkumar, A. (2018). Probabilistic fasttext for multi-sense word embeddings. *arXiv preprint arXiv:1806.02901*.

Babur, Ö. (2019). A labeled ecore metamodel dataset for domain clustering. *Zenodo*.

Babur, Ö., & Cleophas, L. (2017). Using n-grams for the automated clustering of structural models. In *International conference on current trends in theory and practice of informatics.*

Babur, Ö., Cleophas, L., & van den Brand, M. (2016). Hierarchical clustering of metamodels for comparative analysis and visualization. In *European conference on modelling foundations and applications.*

Basciani, F., Di Rocco, J., Di Ruscio, D., Iovino, L., & Pierantonio, A. (2016). Automated clustering of metamodel repositories. In *International conference on advanced information systems engineering.*

Basciani, F., Pierantonio, A., Iovino, L., et al. (2015). A tool for clustering metamodel repositories. In *Model driven engineering languages and systems.*

Behere, S., & Törngren, M. (2016). A functional reference architecture for autonomous driving. *Information and Software Technology*, *73*(C), 136–150.

Blouin, D., & Borde, E. (2020). Aadl: A language to specify the architecture of cyber-physical systems. *Foundations of Multi-Paradigm Modelling for Cyber-Physical Systems*, 209–258.

Cai, H., Zheng, V. W., & Chang, K. C.-C. (2018). A comprehensive survey of graph embedding: Problems, techniques, and applications. *IEEE Transactions on Knowledge and Data Engineering*, *30*(9), 1616–1637.

Clarisó, R., & Cabot, J. (2018). Applying graph kernels to model-driven engineering problems. In *International workshop on machine learning and software engineering in symbiosis.*

Cohen, J. (2013). *Statistical power analysis for the behavioral sciences*.

Duesbury, E., Holliday, J. D., & Willett, P. (2017). Maximum common subgraph isomorphism algorithms. *MATCH Communications in Mathematical and in Computer Chemistry*, *77*(2), 213–232.

Enevoldsen, K., Chung, I., Kerboua, I., Kardos, M., Mathur, A., Stap, D., … others (2025). Mmteb: Massive multilingual text embedding benchmark. *arXiv preprint arXiv:2502.13595*.

Feiler, P. H., & Gluch, D. P. (2012). *Model-based engineering with aadl: an introduction to the sae architecture analysis & design language*.

Feiler, P. H., Gluch, D. P., & Hudak, J. J. (2006). *The architecture analysis & design language (aadl): An introduction* (Tech. Rep.).

Feiler, P. H., & Rugina, A. (2007). *Dependability modeling with the architecture analysis & design language (aadl).*

Gao, X., Xiao, B., Tao, D., & Li, X. (2010). A survey of graph edit distance. *Pattern Analysis and Applications*, *13*(1), 113–129.

Guth, J., Breitenbücher, U., Falkenthal, M., Fremantle, P., Kopp, O., Leymann, F., & Reinfurt, L. (2017). A detailed analysis of iot platform architectures: concepts, similarities, and differences. In *Internet of everything: Algorithms, methodologies, technologies and perspectives.*

Guth, J., Breitenbücher, U., Falkenthal, M., Leymann, F., & Reinfurt, L. (2016). Comparison of iot platform architectures: A field study based on a reference architecture. In *Ieee cloudification of the internet of things.*

Hatcliff, J., Belt, J., Robby, & Carpenter, T. (2021). Hamr: an aadl multi-platform code generation toolset. In *International symposium on leveraging applications of formal methods.*

Hatcliff, J., Belt, J., Robby, Legg, J., Stewart, D., & Carpenter, T. (2023). Automated property-based testing from aadl component contracts. In *International conference on formal methods for industrial critical systems.*

Hatcliff, J., Belt, J., Robby, Legg, J., Stewart, D., & Carpenter, T. (2025). Automated property-based testing from aadl component contracts. *International Journal on Software Tools for Technology Transfer*, 1–28.

Hochgeschwender, N., Biggs, G., & Voos, H. (2018). A reference architecture for deploying component-based robot software and comparison with existing tools. In *IEEE international conference on robotic computing.*

Hugues, J., & Brau, G. (2014). Analysis as a first-class citizen: an application to architecture description languages. In *IEEE international symposium on object/component/service-oriented real-time distributed computing.*

Josey, A., Lankhorst, M., Band, I., Jonkers, H., & Quartel, D. (2016). An introduction to the archimate® 3.0 specification. *White Paper from The Open Group*.

Khalilipour, A., Bozyigit, F., & Challenger, M. (2024). Intelligent model management based on textual and structural extraction-an exploratory study. In *International conference on web research.*

Khalilipour, A., Bozyigit, F., Utku, C., & Challenger, M. (2022). Categorization of the models based on structural information extraction and machine learning. In *International conference on intelligent and fuzzy systems.*

Liu, J., Zeng, J., Wang, X., & Liang, Z. (2023). Learning graph-based code representations for source-level functional similarity detection. In *IEEE/ACM international conference on software engineering.*

Liu, Y., Ott, M., Goyal, N., Du, J., Joshi, M., Chen, D., … Stoyanov, V. (2019). Roberta: A robustly optimized bert pretraining approach. *arXiv preprint arXiv:1907.11692*.

López, J. A. H., & Cuadrado, J. S. (2020). Mar: a structure-based search engine for models. In *ACM/IEEE international conference on model driven engineering languages and systems.*

López, J. A. H., Durá, C., & Cuadrado, J. S. (2023). Word embeddings for model-driven engineering. In *ACM/IEEE international conference on model driven engineering languages and systems.*

López, J. A. H., Rubei, R., Cuadrado, J. S., & Di Ruscio, D. (2022). Machine learning methods for model classification: a comparative study. In *International conference on model driven engineering languages and systems.*

Losavio, F., Ordaz, O., & Esteller, V. (2015). Quality-based bottom-up design of reference architecture applied to healthcare integrated information systems. In *Ieee international conference on research challenges in information science.*

Luan, S., Yang, D., Barnaby, C., Sen, K., & Chandra, S. (2019). Aroma: Code recommendation via structural code search. *Proceedings of the ACM on Programming Languages*, *3*(OOPSLA), 1–28.

Lukasová, A. (1979). Hierarchical agglomerative clustering procedure. *Pattern Recognition*, *11*(5-6), 365–381.

Malavolta, I., Lago, P., Muccini, H., Pelliccione, P., & Tang, A. (2012). What industry needs from architectural languages: A survey. *IEEE Transactions on Software Engineering*, *39*(6), 869–891.

Mkaouar, H., Zalila, B., Hugues, J., & Jmaiel, M. (2020). A formal approach to aadl model-based software engineering. *International Journal on Software Tools for Technology Transfer*, *22*(2), 219–247.

Nakagawa, E. Y., Antonino, P. O., Schnicke, F., Capilla, R., Kuhn, T., & Liggesmeyer, P. (2021). Industry 4.0 reference architectures: State of the art and future trends. *Computers & Industrial Engineering*, *156*, 107241.

Nguyen, P. T., Di Rocco, J., Di Ruscio, D., Pierantonio, A., & Iovino, L. (2019). Automated classification of metamodel repositories: a machine learning approach. In *ACM/IEEE international conference on model driven engineering languages and systems.*

Nguyen, P. T., Di Ruscio, D., Pierantonio, A., Di Rocco, J., & Iovino, L. (2021). Convolutional neural networks for enhanced classification mechanisms of metamodels. *Journal of Systems and Software*, *172*, 110860.

Rubei, R., Di Rocco, J., Di Ruscio, D., Nguyen, P. T., & Pierantonio, A. (2021). A lightweight approach for the automated classification and clustering of metamodels. In *ACM/IEEE international conference on model driven engineering languages and systems companion.*

Rukmono, S. A., Ochoa, L., & Chaudron, M. (2024). Deductive software architecture recovery via chain-of-thought prompting. In *ACM/IEEE international conference on software engineering: New ideas and emerging results.*

Şahin, T., Raulf, C., Kızgın, V., Huth, T., & Vietor, T. (2021). A cross-domain system architecture model of dynamically configurable autonomous vehicles. In *International stuttgarter symposium on automotive and engine technology.*

Salton, G., & Buckley, C. (1988). Term-weighting approaches in automatic text retrieval. *Information Processing & Management*, *24*(5), 513–523.

Sas, C., & Capiluppi, A. (2021). Using structural and semantic information to identify software components. In *IEEE international conference on software analysis, evolution and reengineering.*

Steinley, D. (2004). Properties of the hubert-arable adjusted rand index. *Psychological Methods*.

Stewart, D., Liu, J. J., Cofer, D., Heimdahl, M., Whalen, M. W., & Peterson, M. (2021). Aadl-based safety analysis using formal methods applied to aircraft digital systems. *Reliability Engineering & System Safety*, *213*, 107649.

Tokar, J. L. (2017). A comparison of avionics open system architectures. *ACM SIGAda Ada Letters*, *36*(2), 22–26.

Tran, T. D., Rossi, M. T., Soldati, D., Sonzogni, M., Di Salle, A., Iovino, L., & Mariani, L. (2025). *Structural-semantic clustering for architectural models.* Retrieved from https://github.com/dinhtranthi/AADL-Model-Clustering

Warrens, M. J., & van der Hoef, H. (2020). Understanding the rand index. In *Advanced studies in classification and data science.*

Wilcoxon, F. (1992). Individual comparisons by ranking methods. In *Breakthroughs in statistics: Methodology and distribution* (pp. 196–202).

Xu, X., Ahmad, E., Wang, S., Jin, X., Zhan, B., & Zhan, N. (2025). Modeling and verification of hybrid systems by extending aadl. *ACM Transactions on Software Engineering and Methodology*.

Yu, H., Ma, Y., Gautier, T., Besnard, L., Le Guernic, P., & Talpin, J.-P. (2013). Polychronous modeling, analysis, verification and simulation for timed software architectures. *Journal of Systems Architecture*, *59*(10), 1157–1170.

Zhang, Y., Xu, Z., Liu, C., Chen, H., Sun, J., Qiu, D., & Liu, Y. (2023). Software architecture recovery with information fusion. In *Acm joint european software engineering conference and symposium on the foundations of software engineering.*

Zou, Y., Yang, Z., Liu, H., Liang, J., Zhou, Y., & Gu, Z. (2025). Automated aadl architecture modeling: Leveraging large language models for safety-critical software. In *International conference on model driven engineering languages and systems companion.*

About the authors

Thi Dinh Tran is a PhD student at Gran Sasso Science Institute, Italy. Her research is focused on Natural Language Processing, AI for Code, and Model-Driven Engineering. You can contact the author at thidinh.tran@gssi.it.

Maria Teresa Rossi is a researcher at Gran Sasso Science Institute, Italy. Her research is focused on Model-Driven Engineering, software testing, and software architecture. You can contact the author at mariateresa.rossi@gssi.it.

Davide Soldati is a Master's student at the University of Milano-Bicocca, Italy. His research is focused on software architecture and model-driven engineering.

Mauro Sonzogni is a Master's student at the University of Milano-Bicocca, Italy. His research is focused on software architecture and model-driven engineering.

Amleto Di Salle is an Assistant Professor at Gran Sasso Science Institute. His research activities span several aspects of Software Engineering, including distributed systems composition, software architecture, and model-based software engineering. You can contact the author at amleto.disalle@gssi.it.

Ludovico Iovino is an Associate Professor at Gran Sasso Science Institute, Italy. His research is focused on Model-Driven Engineering, model transformations, and model evolution. You can contact the author at ludovico.iovino@gssi.it.

Leonardo Mariani is a Full Professor at the University of Milano-Bicocca, Italy. His research is focused on software testing, software quality, and the interplay between AI and software engineering. You can contact the author at leonardo.mariani@unimib.it.