Evaluating Software Architecture Diagram Connectivity with Vision Large Language Models

Niccolò Scommegna*
niccolo.scommegna@edu.unifi.it
Università di Firenze - DINFO
Florence, Italy

Luca Bindini*
luca.bindini@unifi.it
Università di Firenze
Florence, Italy

Kimiya Noor Ali
kimiya.noorali@unifi.it
Università di Firenze
Florence, Italy

Roberto Verdecchia
roberto.verdecchia@unifi.it
Università di Firenze
Florence, Italy

Simone Marinai
simone.marinai@unifi.it
Università di Firenze
Florence, Italy

Abstract

Software architecture diagrams communicate software structure and relationships, but automatically estimating their connectivity remains difficult. We study whether Vision Large Language Models, e.g., LLaVA, Gemma, Mistral, and Qwen-VL, can estimate connectivity through a localized proxy: counting arrowheads, usually denoting dependencies, control flows, or communication links. We evaluate local and cloud-based VLLMs on annotated architecture diagrams, showing that the strongest models capture useful connectivity information: Qwen3-VL-Cloud achieves the best overall error-based performance (MAE 1.73, RMSE 3.65), while Gemma4 obtains the highest exact-match accuracy (48.1%). However, performance degrades on large and dense diagrams. To address this limitation, we introduce an image-slicing strategy counting arrowheads in sub-regions; a 2×2 partition improves tolerance-based accuracy from 65.8% to 73.3%. Overall, while estimating diagram connectivity and complexity remains challenging, preliminary results suggest VLLMs combined with simple diagram-aware preprocessing are promising for arrowhead localization and graph reconstruction.

CCS Concepts

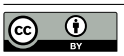
• **Applied computing** → **Graphics recognition and interpretation**; • **Computing methodologies** → *Information extraction.*

Keywords

Vision Large Language Models, Software Architecture, Graphics Recognition and Document Understanding

ACM Reference Format:
Niccolò Scommegna, Luca Bindini, Kimiya Noor Ali, Roberto Verdecchia, and Simone Marinai. 2026. Evaluating Software Architecture Diagram Connectivity with Vision Large Language Models. In *Proceedings of the 2026 ACM Symposium on Document Engineering (DocEng '26), August 25–28, 2026, Fribourg, Switzerland.* ACM, New York, NY, USA, 4 pages. https://doi.org/10.1145/3820755.3832802

*Both authors contributed equally to this research.



This work is licensed under a Creative Commons Attribution 4.0 International License.
DocEng '26, Fribourg, Switzerland
© 2026 Copyright held by the owner/author(s).
ACM ISBN 979-8-4007-2786-3/2026/08
https://doi.org/10.1145/3820755.3832802

1 Introduction

Software architecture documentation is commonly used to describe a system from different perspectives and to support communication among stakeholders. Architectural views can capture structural organization, runtime behavior, deployment decisions, and development concerns [3, 4]. In practice, however, many architecture diagrams found in repositories and project documentation are informal boxes-and-arrows sketches rather than fully standardized models [6]. These diagrams are still valuable because they encode software elements and their relationships, but their interpretation requires combining text recognition, symbol detection, spatial reasoning, and an understanding of diagram conventions.

A key aspect of architectural diagrams is connectivity: arrows and connectors often represent dependencies, e.g., data flows and control flows, communication paths, or deployment relationships. Estimating connectivity can therefore provide useful information regarding the structural complexity of architectural diagrams. However, reconstructing the complete underlying graph is challenging, especially when diagrams contain heterogeneous notation, overlapping elements, small arrowheads, or dense regions. Therefore, we focus on a localized visual proxy for connectivity: counting arrowheads. Although this does not fully recover architectural semantics, it provides a measurable signal related to the number of directed relationships represented in the diagram.

Vision Large Language Models (VLLMs) extend language models with visual input processing and have shown strong performance on multimodal tasks [2, 5, 9]. Their ability to jointly process visual structure and natural language makes them promising candidates for software diagram analysis. At the same time, prior work shows that VLLMs can struggle with precise localization and counting of small repeated objects [7]. This limitation is particularly relevant for architecture diagrams, where arrowheads may be small, visually similar, and distributed across large images.

Recent studies on UML and software architecture diagram understanding suggest that multimodal models can partially extract information contained in architectural diagrams, but still face limitations in structural reasoning and reliable graph reconstruction [1, 8]. This motivates our investigation of whether current VLLMs can estimate diagram connectivity through arrowhead counting, and if simple preprocessing strategies can improve their performance on larger and denser diagrams.

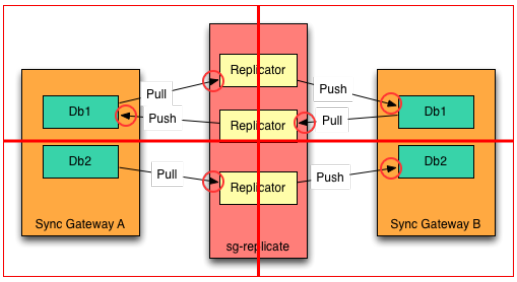


Figure 1: Architectural diagram example. Arrowheads are circled, red lines indicate the 2×2 patch partition.

2 Method

2.1 Problem Definition

Architectural diagrams represent software systems through components and directed relationships. In this work, we approximate diagram connectivity by counting visible arrowheads, which provide localized visual markers of directed relationships and avoid the need to reconstruct the full underlying graph. Although more localized than complete connections, arrowhead counting remains challenging due to variations in image quality, resolution, graphical style, visual density, and overlapping arrowheads. These characteristics make the task a useful benchmark for evaluating counting and visual reasoning of modern VLLMs. Formally, given an image I, our objective is to estimate the total number of visible arrowheads y. A VLLM produces a prediction $\hat{y}$, which is compared against a manually annotated reference value. In this work, an *arrowhead* is defined as the graphical termination of a solid or dashed line, regardless of its geometric shape, for example, triangular or hollow.

2.2 Preprocessing and Patch-Based Counting

The images used in this study were obtained from public GitHub repositories through the dataset released by Migliorini et al. [6]. Before inference, a preprocessing stage was applied to standardize image formats and reduce the impact of potential visual artifacts.

In particular, for images containing an alpha channel, transparent pixels were converted to white. This choice was motivated by the observation that, in most cases, transparency corresponds to the image background. Replacing transparent pixels with white ones helps preserve the readability of the visual content and avoids biases arising from the way transparency is handled by different models. Several models tend to interpret transparent pixels as black pixels, potentially reducing the distinguishability between the background and graphical elements and causing the loss of relevant details in images where arrows and connections are represented in black.

To improve counting performance on large images, we additionally adopted an inference strategy based on image partitioning. As illustrated in Figure 1, each image was first divided into a regular 2×2 grid, yielding four non-overlapping regions of equal size. In the second configuration, each of the four regions was further subdivided into a 2×2 grid, resulting in an overall 4×4 partition composed of sixteen patches. In both configurations, each patch was processed independently using the same prompt employed for the full image. The final count was then obtained by aggregating the predictions generated for the individual regions. This strategy aims to increase the effective resolution of the elements observed by the model, facilitating the identification of small arrowheads and improving counting performance in high-resolution images containing a large number of visual elements.

3 Experiments

We evaluate arrowhead counting in three steps. First, we compare several VLLMs on an annotated benchmark of software architecture diagrams. Second, we analyze the effect of the generation temperature on the most competitive local model. Third, we study if preprocessing and patch-based inference improve robustness when moving to a larger and more heterogeneous diagram collection.

We use two annotated datasets. The first dataset contains 337 software architecture diagrams and is used for the initial comparison across models and for the temperature analysis. These diagrams come from the publicly available corpus of architecture diagrams collected from open-source projects [6]. The second dataset contains 374 diagrams sampled from a larger GitHub corpus. This second set was manually annotated for the number of visible arrowheads and was designed to include a broader range of image sizes and visual densities. To avoid evaluating only simple diagrams, the sample was selected via a bin-based strategy by preserving diagrams of different size and complexity distribution portions.

All diagrams were processed using the same zero-shot prompt. The exact prompt is reported in Listing 1. For each input image, the model was asked to return only the number of arrowheads visible in the diagram. We ran five independent inferences and used the median prediction as output to reduce the impact of occasional unstable generations while preserving a simple inference protocol.

Listing 1: Prompt used for arrowhead counting.

```
Task: Arrowhead Counting in Software Architecture Diagrams
Analyze the provided diagram and count the total number of
      individual arrowheads (the tips of the arrows).
Rules:
1. Count every single arrowhead as 1. (e.g., if a single line has
      an arrowhead at both ends, count it as 2).
2. A valid arrowhead MUST be attached to a segment (either a solid
       or dashed line).
3. Arrowheads can appear in various visual styles: open (v-shaped)
      , closed (hollow triangles), or filled (solid triangles).
4. Ignore plain lines, line intersections, or floating triangles
      that are not connected to a line.
Reasoning (Internal):
- Scan the diagram for all line connections and terminations.
- Verify if the termination contains a valid arrowhead based on
      the visual styles mentioned.
- Count each identified arrowhead.
Output Instructions:
- Provide ONLY the final total number as an integer.
- Do not provide explanations, text, or punctuation.
Final Answer:
```

The evaluated models are local and cloud-based VLLMs: LLaVA 7B, Gemma3 12B, Mistral-Small3.2 24B, Gemma4 31B, and Qwen3-VL-Cloud 235B. Local models were evaluated with the same inference pipeline, while the cloud one *via* its remote interface.

We report two error-based metrics and three accuracy-based metrics. Given the manually annotated count y_i and the predicted count $\hat{y}_i$, MAE measures the average absolute error and RMSE penalizes larger errors more strongly. We also report exact match accuracy and two dynamic tolerance accuracies. A prediction is

Table 1: Evaluation metrics on arrowhead counting.

Model	MAE	RMSE	Accuracy (%)		
			Exact	Tol. 10%	Tol. 15%
LLaVA (7B)	5.29	7.67	6.18	21.47	25.88
Gemma3 (12B)	13.21	16.05	0.30	3.86	6.53
Mistral-Small3.2 (24B)	14.74	22.30	4.45	8.01	8.61
Gemma4 (31B)	1.88	4.30	**48.07**	69.14	73.59
Qwen3-VL-Cloud (235B)	**1.73**	**3.65**	38.82	**72.35**	**77.35**

Table 2: Effect of replacing transparent pixels with white pixels before inference using Gemma4 31B.

Dataset	Setting	Exact	Tol. 10%	Tol. 15%
Transparent images	Original	30.1	43.8	46.6
	Pre-processed	**46.6**	**58.9**	**61.6**
Complete dataset	Original	35.8	55.9	62.8
	Pre-processed	**39.0**	**58.9**	**65.8**

considered correct under tolerance τ when it falls within $\pm\tau$ of the ground-truth count. We use $\tau = 10\%$ and $\tau = 15\%$.

3.1 Model and Temperature Analysis

Table 1 reports the model comparison results. The best overall performance is obtained by Qwen3-VL-Cloud 235B, which reaches the lowest MAE and RMSE, and the highest dynamic accuracies at both 10% and 15% tolerance, indicating that the larger cloud model is the most reliable when approximate counting is acceptable.

Gemma4 31B achieves the highest exact match accuracy. Its MAE and RMSE are only slightly worse than those of Qwen3-VL-Cloud, while its exact match accuracy is substantially higher. This makes Gemma4 31B a strong practical candidate for the remaining experiments: it is competitive with the cloud model, but can be evaluated locally supporting larger-scale experimentation at lower cost.

The smaller models show a clear performance gap. LLaVA 7B obtains moderate results, but remains far from the two best-performing models. Gemma3 12B and Mistral-Small3.2 24B perform poorly, especially in terms of MAE and RMSE. These results confirm that arrowhead counting is not a simple visual question-answering task: it requires precise recognition of many small and repeated graphical elements, and performance varies substantially across VLLMs.

As generative models may produce different answers for the same image, we also evaluate sampling temperature effects. We conduct this analysis using Gemma4 31B and a range of temperature values, from deterministic to more stochastic settings, assessing whether more constrained decoding are beneficial for our task. The model results relatively stable w.r.t. temperature, so we adopt the default setting (temperature = 1.0) in the subsequent experiments.

3.2 Preprocessing and Patch-Based Inference

To avoid issues related to the handling of transparent backgrounds by some VLLMs, we applied the preprocessing step described in Section 2.2. Table 2 reports its impact on the second dataset. On the subset of 73 images with transparency, replacing the transparent background with white improves exact accuracy from 30.1% to 46.6%, dynamic accuracy at 10% tolerance from 43.8% to 58.9%, and dynamic accuracy at 15% tolerance from 46.6% to 61.6%. When the same correction is integrated into the full pipeline and evaluated on all 374 diagrams, the improvement is smaller but still consistent across all metrics. This confirms that transparency is a localized issue affecting only part of the dataset, but also that correcting it is necessary to avoid avoidable counting errors.

The model comparison shows that the strongest VLLMs can count arrowheads with reasonable accuracy on the first annotated dataset. However, errors increase when diagrams become visually dense, when arrowheads are small, or when many connectors are present in the same image. This behavior motivates the patch-based strategy introduced in the method section.

Figure 2 compares full-image inference with patch-based inference on the second annotated dataset, grouped by image-size percentiles. The 2×2 strategy improves average bin-level dynamic accuracy at 15% tolerance from approximately 65.8% to 73.3%. The improvement is particularly visible in middle and large image bins, where the full-image strategy is more likely to miss small arrowheads. This supports the intuition of splitting a large diagram into four regions to increase effective visual resolution available to the model. Figure 2 also reports a 4×4 strategy. Unlike the 2×2 configuration, the finer partitioning consistently underperforms, with an average bin-level accuracy of $\approx$50.0%. This suggests that excessive slicing removes too much context and increases the risk of splitting connections or arrowheads across patch boundaries. Therefore, the 2×2 configuration provides the best trade-off between local detail and global context among the considered strategies.

To better understand the behavior of the selected strategy, Figure 3 compares the manually annotated counts with the predictions obtained using Gemma4 31B and the 2×2 patch-based pipeline. Most low-count diagrams are close to the ideal prediction line, i.e., the model is reliable when diagrams contain few arrowheads. As the number of arrowheads increases the dispersion becomes larger and the model tends to underestimate highly connected diagrams, confirming that arrowhead counting remains challenging for dense architectures, even when patching is used. The scatter plot also clarifies why tolerance-based metrics are important. Exact match is a strict measure and is informative for simple diagrams, but on dense diagrams even small relative errors may correspond to several arrowheads. Dynamic tolerance provides a complementary view of performance, especially when the goal is to estimate diagram connectivity rather than recovering exact graphical annotations.

The experiments lead to three main observations. First, VLLMs are able to capture useful information about the connectivity of software architecture diagrams, but performance depends strongly on the selected model. Second, preprocessing matters: seemingly minor image-format issues, such as transparency, can produce systematic visual artifacts that affect counting. Third, patch-based inference improves robustness on larger and denser diagrams, but only when the patches remain large enough to preserve local context.

4 Conclusion

In this paper, we investigated the ability of VLLMs to estimate the connectivity of software architecture diagrams through arrowhead

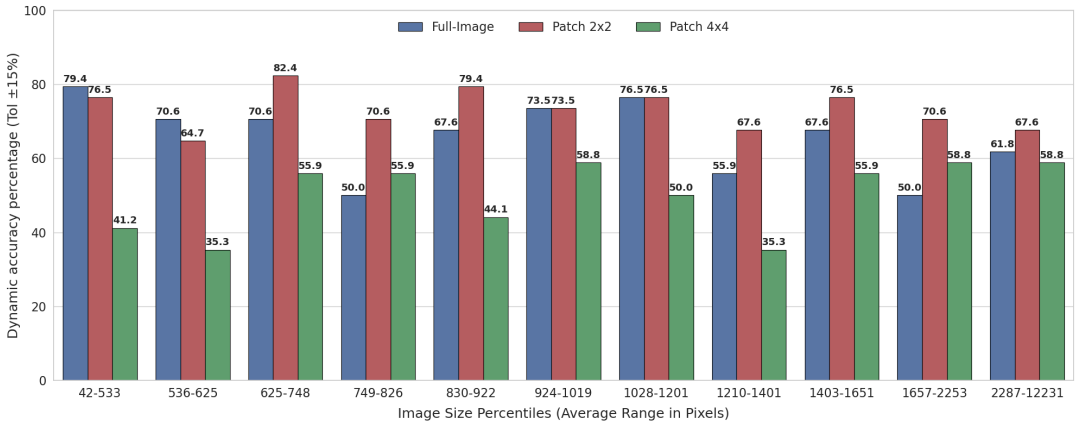


Figure 2: Dynamic accuracy at 15% tolerance on the second annotated dataset, grouped by image-size percentiles. Full-image inference is compared with 2×2 and 4×4 patch-based inference.

Figure 3: Ground-truth arrowhead counts against Gemma4 31B predictions using the 2×2 patch-based strategy.

counting. By considering arrowheads as localized visual proxies for directed relationships, we evaluated if current VLLMs can support the automatic analysis of architectural diagram complexity. Our results show that VLLMs can capture useful diagram connectivity information, although their performance strongly depends on the selected model. Larger and more capable models achieve substantially better results, while smaller models struggle. Experiments highlight that image preprocessing is paramount: minor issues, such as transparent backgrounds, often introduce visual artifacts that negatively affect predictions. Finally, simple patch-based strategy result to improve counting performance on larger and denser diagrams. In particular, splitting the image into a 2×2 grid provides a better trade-off between preserving local context and increasing the effective resolution of small arrowheads. However, the remaining errors on highly connected diagrams indicate that accurate diagram understanding is still challenging for VLLMs.

In conclusion, our study demonstrates that VLLMs represent a viable tool for the efficient and effective interpretation of architectural diagrams. Future work should build on these preliminary findings, working toward the broader vision of fully automated architectural view interpretation, e.g. by considering architectural layers, architectures, and technology stacks, a capability from which both software practitioners and automated software system interpretation can benefit.

The complete annotated dataset of 13,092 software architecture images is publicly available.[1] Future work will investigate partially overlapping patches with duplicate-count suppression, prompt robustness, box counting, and explicit arrowhead localization.

References

[1] Gábor Antal, Richárd Vozár, and Rudolf Ferenc. 2024. Toward a New Era of Rapid Development: Assessing GPT-4-Vision's Capabilities in UML-Based Code Generation. In *LLM4CODE@ICSE*. 84–87. doi:10.1145/3643795.3648391

[2] Shuai Bai, Keqin Chen, Xuejing Liu, Jialin Wang, Wenbin Ge, Sibo Song, Kai Dang, Peng Wang, Shijie Wang, Jun Tang, Humen Zhong, Yuanzhi Zhu, Ming-Hsuan Yang, Zhaohai Li, Jianqiang Wan, Pengfei Wang, Wei Ding, Zheren Fu, Yiheng Xu, Jiabo Ye, Xi Zhang, Tianbao Xie, Zesen Cheng, Hang Zhang, Zhibo Yang, Haiyang Xu, and Junyang Lin. 2025. Qwen2.5-VL Technical Report. *CoRR* abs/2502.13923 (2025). arXiv:2502.13923 doi:10.48550/ARXIV.2502.13923

[3] Paul C. Clements, David Garlan, Reed Little, Robert L. Nord, and Judith A. Stafford. 2003. Documenting Software Architectures: Views and Beyond. In *International Conference on Software Engineering*, Lori A. Clarke, Laurie Dillon, and Walter F. Tichy (Eds.). IEEE Computer Society, 740–741. doi:10.1109/ICSE.2003.1201264

[4] Philippe Kruchten. 1995. The 4+1 View Model of Architecture. *IEEE Softw.* 12, 6 (1995), 42–50. doi:10.1109/52.469759

[5] Haotian Liu, Chunyuan Li, Qingyang Wu, and Yong Jae Lee. 2023. Visual Instruction Tuning. In *Advances in Neural Information Processing Systems*. http://papers.nips.cc/paper_files/paper/2023/hash/6dcf277ea32ce3288914faf369fe6de0-Abstract-Conference.html

[6] Sofia Migliorini, Roberto Verdecchia, Ivano Malavolta, Patricia Lago, and Enrico Vicario. 2024. Architectural views: the state of practice in open-source software projects. In *European Conference on Software Architecture*. Springer, 396–415.

[7] Valeria Nardoni, Kimiya Noor Ali, Zahra Ziran, and Simone Marinai. 2025. Visual Large Language Models for Graphics Understanding: A Case Study on Floorplan Images. In *ACM Symposium on Document Engineering, DocEng 2025, Nottingham, UK, September 2-5, 2025*, Steven R. Bagley, Steven J. Simske, Charlotte Curtis, and Cerstin Mahlow (Eds.). ACM, 29:1–29:4. doi:10.1145/3704268.3748681

[8] Shuyin Ouyang, Jie M. Zhang, Jingzhi Gong, Gunel Jahangirova, Mohammad Reza Mousavi, Jack Johns, Beum Seuk Lee, Adam Ziolkowski, Botond Virginas, and Joost Noppen. 2026. Benchmarking and Evaluating VLMs for Software Architecture Diagram Understanding. *CoRR* (2026). https://doi.org/10.48550/arXiv.2604.04009

[9] Gemma Team. 2024. Gemma: Open Models Based on Gemini Research and Technology. *CoRR* abs/2403.08295 (2024). arXiv:2403.08295 doi:10.48550/ARXIV.2403.08295

[1] https://doi.org/10.5281/zenodo.21398106