Paper2SysArch: Structure-Constrained System Architecture Generation from Scientific Papers

Supplementary Material

S1. Detailed Workflow and Agent Prompts

S1.1. P2SA Agent

Complementing the conceptual overview provided in the main paper, this section details the **formal definitions** and **implementation specifics** of the Paper2SysArch Agent. Specifically, we first provide the rigorous data schema and topological constraints for the Hierarchical Graph Representation. Subsequently, we expand the Automated Generation Pipeline into a granular, seven-step **Neuro-Symbolic** workflow, elaborating on the specific mechanisms for distributed drafting, topological regularization, and content-aware layout optimization.

S1.1.1. Hierarchical Graph Representation of SysArch

To structurally represent the system architecture(which is essential for our Structure-Constrained generation), we define a **Three-level Hierarchical Graph**, abstracting the system from macroscopic modules to microscopic components. Here is an example in Listing 1.

1. Hierarchical Levels. The graph consists of three distinct layers:

- **Module Level (L_1):** Represents the top-level abstraction of the system, corresponding to major processing phases such as Intent Recognition, Data Preprocessing, or Model Inference.
- **Entity Level (L_2):** Comprises **Tool** and **Data** objects. Tools refer to functional units (e.g., Transformer blocks, external libraries, or specific algorithms), while Data represents I/O streams and intermediate features (e.g., feature maps, vectors, or raw text).
- **Component Level (L_3):** Constitutes the atomic visualization primitives within each object. We define three types of components: (1) *Icon*: Functional identifiers (e.g., logos); (2) *Text*: Descriptive strings (e.g., algorithm names, parameter descriptions); and (3) *Image*: Pixel-level visuals (e.g., statistical charts or sample inputs).

2. Data Schema and Constraints. We serialize the graph structure using the JSON format. For any node N, the attributes are defined as follows:

- **Metadata Attributes:** The `type` field specifies the node category, where `type` $\in$ {`module`, `tool`, `component-icon/text/image`}. Each node is assigned a globally unique identifier (`id`) and a semantic label (`name`). Note that while `name` can be reused across nodes, `id` must be unique.
- **Recursive Hierarchy:** The `children` field defines the nesting structure. For L_1 and L_2 nodes, this field contains a list of child objects. For L_3 leaf nodes (components), this field stores the specific content payload (e.g., a text string, icon description, or image path).
- **Topological Connections:** The `edges` field describes the logical data flow. To ensure modular decoupling, directed edges are strictly constrained to connect **sibling nodes** under the same parent node. Each edge is defined as a tuple $\{id, name, sources, targets\}$, representing a directional connection from the source node(s) to the target node(s).

S1.1.2. Phase I: Semantic Parsing and Macro-Planning

This phase aims to translate unstructured academic texts into a structured top-level topology. It involves two specific steps:

- **Step 1: Information Extraction.** The system deploys an *Analyst Agent* to comprehend the input paper. The agent is instructed to extract core information across five dimensions: (1) overall task goals; (2) major modules and their responsibilities; (3) critical data flows (inputs/outputs/intermediate representations); (4) key algorithms or models; and (5) system constraints. This information is compressed into a structured system summary, serving as the contextual foundation for subsequent generation.
- **Step 2: Coarse-grained Topology Construction.** Based on the extracted summary, an *Architect Agent* constructs the skeletal graph. This step instantiates only the Root Node and the first-level (L_1) Module Nodes, establishing the macroscopic boundaries and directed connections between major modules without delving into internal details.

S1.1.3. Phase II: Distributed Generation and Global Alignment

To handle the generation of complex system details, we adopt a "Divide-and-Conquer" strategy that transitions from parallel drafting to sequential refinement:

- **Step 3: Parallel Sub-graph Instantiation.** To ensure context isolation and generation efficiency, the system assigns an independent *Designer Agent* to each L_1 module. These agents work in parallel to populate the internal L_2 Entity Level (Tools/Data) and L_3 Component Level. This phase generates local sub-graphs (G_{sub}) that guarantee local logical validity.

```
{
    "type": "module",
    "id": "n1",
    "name": "Data Preprocessing Module",
    "children": [
        {
            "type": "tool",
            "id": "n2",
            "name": "Data Cleaning Tool",
            "children": [
                {
                    "type": "component-text",
                    "id": "n3",
                    "name": "Algorithm Name",
                    "children": "Outlier Detection&Handling"
                },
                {
                    "type": "component-icon",
                    "id": "n4",
                    "name": "Cleaning Icon",
                    "children": "broom_trash_icon.png"
                }
            ],
            "edges": [
                {
                  "sources": ["n2"],
                  "targets": ["n5"],
                  "id": "e1",
                  "name": "Cleaned Data"
                }
            ]
        },
        {
            "type": "tool",
            "id": "n5",
            "name": "Data Standardization Tool",
            "children": [
                {
                    "type": "component-text",
                    "id": "n6",
                    "name": "Method",
                    "children": "Z-score Standardization"
                },
                {
                    "type": "component-image",
                    "id": "n7",
                    "name": "Example Chart",
                    "children": "path_to/image.png"
                }
            ]
        }
    ],
    "edges": [
        {
          "sources": ["n1"],
          "targets": ["n5"],
          "id": "e2",
          "name": "Standardized Data"
        }
    ]
}
```


Listing 1. Example of the Graph JSON Data Structure.

- **Step 4: Context-Aware Sequential Refinement.** To address global consistency issues arising from parallel generation (e.g., ID conflicts or mismatched interfaces across modules), the system enters a serial revision mode. Local drafts are aggregated into a *Global Context*. Each Designer Agent is sequentially activated to review and refine its responsible sub-graph under the global view, ensuring unique IDs and semantic alignment of I/O interfaces between modules.

S1.1.4. Phase III: Topological Regularization

This phase enforces the constraints defined in the Hierarchical Graph Protocol using a **Neuro-Symbolic** approach:

- **Step 5: Hybrid Constraint Enforcement.** According to the protocol, directed edges are strictly prohibited from crossing module boundaries to connect internal components directly (preserving encapsulation). We employ a dual-check mechanism:
  1. **Semantic Filtering (Neuro):** An LLM first reasons over the graph to identify and reroute obvious logical errors in cross-module connections.
  2. **Symbolic Pruning (Symbolic):** A deterministic program traverses all edges, strictly deleting any residual connections that violate topological constraints, ensuring the final graph structure is structurally legal.

S1.1.5. Phase IV: Adaptive Layout and Multi-modal Rendering

The final phase transforms the logical graph structure into visually aesthetic and editable documents:

- **Step 6: Content-Aware Layout Optimization.** The system first dynamically calculates the initial size weights of nodes based on the length of their internal text content. Subsequently, it invokes the RectPacking algorithm from the **Eclipse Layout Kernel (ELK)**. This algorithm optimizes the geometric coordinates (x, y, w, h) for each node, minimizing the canvas area while achieving a compact, non-overlapping layout.
- **Step 7: Generative Rendering and Vector Compilation.** The system utilizes a Text-to-Image model to synthesize matching icons for each component. Finally, the `python-pptx` engine compiles all geometric information, text content, and image assets into editable PowerPoint slides and high-resolution images, completing the end-to-end generation.

S1.1.6. Prompt Engineering Details

To enable the LLM to generate the specific hierarchical graph structure defined in our protocol, we designed a comprehensive system prompt. The prompt explicitly defines the role, the three-layer structure, and the strict JSON schema constraints. The full prompt is provided below in Figure 8

S1.2. P2SA Benchmark Construction

To construct the Paper2SysArch Benchmark described the main paper, we implemented a standardized data collection and filtering pipeline. This section details the specific tools and parameters used.

Prompt for Paper Information Extraction

< System >

```
### Role:
You are an Expert in Understanding Academic System
Architectures.

### Task Description
Given the original paper text (or excerpts), please
summarize the following aspects point-by-point:
• System Name
• Task & Overall Goal
• Main Modules/Stages & Responsibilities
• Key Data/Information Flow (Input → Process →
  Output)
• Core Models/Algorithms & Their Roles
• Key Constraints or Assumptions (if applicable)

### Requirements

• Language should be concise and well-organized.
• The structure must facilitate the subsequent
  conversion into a system architecture diagram.
```

< User >

```
Please analyze the following paper text.
Paper Text: {paper_content}
```

Figure 7. The detailed Prompts used in SystemUnderstandAgent

S1.2.1. Data Parsing and Extraction

We sourced papers from top-tier AI conferences (e.g., CVPR, ICCV, NeurIPS, ICLR) over the past five years. To handle the unstructured PDF data, we employed the **PyMuPDF** library:

- **Image Extraction:** The pipeline iterates through PDF pages to extract all embedded bitmap resources, saving them as PNG files.
- **Text Association:** We extracted the full text and established page-level mapping between images and text. Specifically, the *Abstract* and *Figure Captions* were isolated to serve as contextual grounding for the subsequent filtering step.

S1.2.2. VLM-based Filtering and Cleaning

To filter out non-architectural images (e.g., performance plots or qualitative results) and retain only genuine System Architecture Diagrams, we deployed a VLM-based discriminator:

- **Discrimination Mechanism:** The extracted PNG images and the paper's abstract are fed into a VLM. The model is prompted to determine whether the image depicts the core system architecture or methodological pipeline of the paper.
- **Thresholding:** The model outputs a confidence score. We set a strict confidence threshold of **0.75**. Only images scoring above this value are candidates for retention.
- **Deduplication:** To ensure a one-to-one mapping for supervised training, we retained only the single highest-scoring architecture diagram for each paper.

Ultimately, combined with the manually curated test set, we established a dataset comprising 3,000 samples.

S1.3. P2SA Auto Evaluation System

This section provides the algorithmic implementation details and hyperparameter settings for the three-tier evaluation framework proposed in Sec. 3.4 of the main paper.

S1.3.1. Specialized Evaluation Agents

To address the issues of attentional dispersion and inconsistent instruction following inherent in general-purpose Vision-Language Models (VLMs) when processing information-dense diagrams, we propose a **Role-Based Multi-Agent Evaluation Framework**.

Instead of a coarse "one-prompt-fits-all" approach, we adopt the **"VLM-as-an-Agent"** paradigm. We decompose the complex task of system architecture evaluation into a series of atomic sub-tasks. Each agent represents a functional **encapsulation** of the VLM—constraining its broad multimodal capabilities into specialized expert discriminators via specific system prompts.

Specifically, we define five core agents to conduct orthogonal evaluations across structural, semantic, visual, and layout dimensions:

GraphExtractAgent (Topological Parsing Specialist): This agent is responsible for abstracting the diagram's structure. It is instructed to ignore specific visual styles and focus solely on identifying nodes (entities) and edges (relations). It converts unstructured image pixels into a structured GraphJSON representation, providing a machine-readable intermediate format for subsequent semantic consistency calculations.

IconExamineAgent (Visual-Semantic Alignment Specialist): This agent focuses on the alignment between visual elements and textual labels. It receives module definitions from the GraphJSON and locates the corresponding icon regions. Its core task is to judge whether the *visual metaphor* of an icon accurately conveys the functional semantics of the module (e.g., checking if a "Database" module uses a storage bucket icon), thereby quantifying visual accuracy.

LayoutExamineAgent (Geometric Layout Auditor): Acting as a visual QA specialist, this agent is dedicated to detecting geometric defects. It is specifically prompted to scan the image for "violation patterns," such as unnecessary line crossings, text overflowing bounding boxes, and overlapping elements, to calculate layout rationality penalties.

Prompt for Graph Generation

<System>

Role:
You are an expert Diagram Designer responsible for hierarchically designing and planning a system data flow diagram based on the given description text.

Design Rules
The diagram follows a three-level hierarchical structure:
- **Top Level (Module Layer):** Represents the overall system modules or phases (e.g., Intent Recognition, Data Preprocessing, Model Inference).
- **Middle Level (Tool/Data Layer):**
  - **Tool Objects:** Tools utilized within each module (e.g., GRU, Transformer, LLM interfaces).
  - **Data Objects:** Inputs, outputs, and intermediate results (e.g., feature maps, vectors).
- **Bottom Level (Component Layer):** Concrete content contained within each module/tool/data object.

Directed edges can exist between two nodes belonging to the same parent node, determined by the data flow direction.

The Component Layer includes 3 types of components:
1. **Icon Component (icon):** Icons assisting in explaining functions (e.g., LOGO).
2. **Text Component (text):** Explanatory text (e.g., tool names, descriptions, I/O examples).
3. **Image Component (image):** User-input images (e.g., charts).

Design Requirements
- Focus primarily on the **logical layout**; do not consider spatial layout.
- Both Module and Tool Layers can directly contain components.
- For Component Layer nodes, include specific content: description for icons, body text for text, and file path for images.
- Ideally, every module should have an icon for illustration.

Output Format
Output in JSON format.

Field Regulations
For an object:
- Use type to distinguish hierarchy: "module", "tool", "component-text/icon/image".
- Use id for a globally unique identifier string (e.g., "n1").
- Use name for the object's name.
- Use children to list child nodes. For Component Layer, it is a **single string** (content). For others, it is a list of objects.
- Use edges for connections **<between child objects>**.
  - Format: {"sources": ["id"], "targets": ["id"], "id": "e_id", "name": "label"}.
  - sources and targets are single-element lists.
  - Edges can only exist between objects under the same direct parent.

Example:
(See Listing 1 for the JSON schema example.)

<User> in Top-Level Design(Step2)

Please design only the Top-Level Module Layer: Return a single JSON object representing the root object (type="module"). Its children must contain only Module Layer nodes (type="module"), along with the edges connecting these modules. Do not design Tool or Component Layers; strictly adhere to the aforementioned Output Format and Field Regulations.
{User Input}

<User> in Sub-graph Design(Step 3 and 4)

Please perform sub-level design for the specified module. Generate only the Tool Layer (type="tool") and Component Layer (component-*) within this module. You may include edges to represent connections between child objects. Maintain the module ID; do not create cross-module edges. Strictly adhere to the output format.

- Target Module:**{module_id}**
- Original Requirement: **{User Input}**
- Context:
  - "top_design": **{top design result in Step 2}**,
  - "other_modules": **{sibling designs of Step 3 (is empty during Step 3)}**,
  - "revision_requirements": **{extra revision requirements given during Step 4}**,

Output Constraint: Return a single JSON object where type="module" and id matches the target module, containing its populated children and (optional) edges.

Figure 8. The detailed Prompts used to guide the LLM in generating structurally valid hierarchical graphs.

SystemUnderstandAgent (Cognitive Simulator): This agent simulates the high-level cognitive process of a *human reader*. By combining the paper abstract (context) with the input image, it generates a natural language description of the system workflow. By comparing the agent's understanding of the "Generated Diagram" versus the "Ground Truth," we can assess the *information fidelity* at a macroscopic level.

TextLegibilityAgent (Low-level Perception Specialist): Given that generative models often produce text artifacts, this agent is tasked with low-level visual perception. It focuses on detecting character distortion, blurring, or unreadable resolution, ensuring the usability of the diagram at the detail level.

Through this **divide-and-conquer** strategy, we focus the VLM's attention on single dimensions, significantly enhancing the robustness and interpretability of the automated evaluation.

S1.3.2. Semantic Layer Evaluation Details

The assessment of semantic consistency relies on the accurate alignment between nodes in the Generated Graph (N_g) and the Ground Truth Graph (N_{gt}).

1. Node Matching Algorithm. As defined in Eq. (1) of the main paper, the matching score aggregates multiple similarity features. In our implementation:

- **Text Feature:** We utilize the **CLIP model** to compute the semantic cosine similarity between node labels.
- **Two-Stage Strategy:**
  - *Stage 1 (Anchor Matching):* This stage prioritizes CLIP-based text similarity and degree similarity to identify high-confidence anchor pairs.
  - *Stage 2 (Structure Propagation):* Building upon the anchors, we increase the weights of structural features (i.e., Neighbor Similarity and Ancestor Chain Similarity). This allows the system to recall nodes that perform identical structural roles but use paraphrased textual descriptions.

2. Scoring Mechanism. Instead of simple binary classification (Precision/Recall), we calculate a continuous **similarity score**:

- **Node Consistency:** The average semantic similarity score of all matched node pairs.
- **Edge and Hierarchy Consistency:** Based on the aligned nodes, we calculate the similarity scores for edges (connectivity) and hierarchical relationships (parent-child containment) preserved in the generated graph.

S1.3.3. Layout Layer Evaluation Details

Unlike rule-based SVG parsing, our layout evaluation is entirely **vision-based**, leveraging the perceptual capabilities of VLMs.

- **Mechanism:** The *LayoutExamineAgent* (a VLM) directly analyzes the generated PNG image to identify visual defects.
- **Penalty System:** The score starts at a perfect 1.0. For each detected instance of the following defects, **0.1 points** are deducted:
  - *Line Crossings:* Connection lines intersecting in non-node areas.
  - *Element Overlaps:* Unintended occlusion between nodes, icons, or text boxes.
  - *Text Overflows:* Text content extending beyond its container boundaries.

S1.3.4. Visual Layer Evaluation Details

This layer assesses aesthetics and information fidelity, also driven by VLM agents:

- **Icon Relevance:** We do not employ separate segmentation models. Instead, the *IconExamineAgent* (VLM) directly analyzes local image regions containing the icon and module name. It evaluates whether the visual metaphor of the icon accurately reflects the functional semantics of the module (e.g., checking if a "Data Cleaning" module uses a broom or filter icon).
- **System Understandability:** This measures information loss via semantic reconstruction. The VLM generates textual captions for both the generated diagram and the ground truth. We then calculate the semantic similarity between these two captions.
- **Text Legibility:** The *TextLegibilityAgent* scans the image for artifacts specific to generative models, such as blurred, distorted, or unreadable characters.

S1.3.5. Prompt Engineering for Evaluation Agents

To instantiate the specialized evaluation capabilities of the VLM, we designed distinct system prompts for each of the five agents. These prompts explicitly define the agent's **persona** (e.g., "Geometric Layout Auditor" or "Cognitive Simulator") and outline the rigorous **adjudication criteria** (e.g., specific penalties for line crossings or blurriness). By conditioning the VLM with these structured instructions, we transform a general-purpose model into a suite of focused expert judges. The full system prompts for all evaluation agents are provided in Figures 9-13

Prompt for GraphExtractAgent

< System >

Role:
You are a Graph Processing Engineer responsible for extracting the topological structure from an input system architecture diagram and its description.

Task Description
First, comprehend the image description. Then, based on your understanding of the diagram, deconstruct the input system architecture to extract its essence. Abstract it into a graph structure composed of nodes and edges, and represent it in JSON.

Requirements

- Any semantic entity with independent content can be a node (e.g., a function, processing step, or I/O result). Do **not** over-split; for instance, a single sentence usually does not constitute a node. Content on lines is generally considered edge labels.
- **Granularity Control:** Node granularity should not be too fine; maintain consistency with the granularity of the **Image Description**!
- **Containment vs. Connection:**
  - If one node spatially encompasses another, it is a **containment** (parent-child) relationship.
  - If nodes are linked by arrows, it is a **connection** relationship.
- Judgment must rely on **semantics**, not just spatial layout.

Representation Rules
Top-level fields: graph and explain.
1. graph: Contains nodes and edges.
   - nodes: List of objects. Each has id (unique), name (summarized by you), and children (list of child IDs; empty if leaf).
   - edges: List of objects. Each has source (id), target (id), and name (summarized by you).
2. explain: Detailed explanation text.
   - Explain your understanding of the system's principles and purpose.
   - Explain the content/function of each node.
   - Explain the content/data flow of each edge.

Output Format
Output in JSON format wrapped in ```json ```.

Example:
```json{ "graph": { "nodes": [{ "id": "n0", "name": "Input Data", "children": [] }, ...], "edges": [{"id": "e1", "source": "n0", "target": "n2", "name": "Data Input"}, ...] }, "explain": "..." }```

< User >
Please extract the graph structure based on the image content and the paper text.
Paper Text: **{paper_text}**

Figure 9. The detailed Prompts used in GraphExtractAgent

Prompt for IconExamineAgent

< System >
Role:
You are an Icon Inspection Engineer with a deep understanding of icon design and visual representation in system architecture diagrams.

Task Description
Referencing the original paper text and the provided graph structure (nodes and edges), examine the icon for each module in the input image.
- First, determine if a module has an icon.
- If **no icon** exists, return an empty string "".
- If an **icon exists**, return a textual description of the icon.

Requirements

- Your description must focus on the **visual appearance** of the icon itself.
- Do **not** be biased by the module's textual content or function name. The module information is provided **solely** to help you locate the module within the image, not to dictate your description of the icon.

Output Format
Output in JSON format wrapped in "`json, containing no other text.
- **Keys:** The id of each node in the graph structure.
- **Values:** The textual description of the icon.

Example:
```json{ "n1": "A sophisticated robot typing text", "n2": "A Python language logo", "n3": "A database cylinder symbol" } ```

< User >

Please inspect the icons within the image based on the visual content, the paper text, and the graph structure.

Paper Text: **{desc}**
Graph Structure: **{graph}**

Figure 10. The detailed Prompts used in IconExamineAgent

Prompt for LayoutExamineAgent

< System >

Role:
You are a Diagram Layout Inspection Engineer capable of **meticulously** detecting layout anomalies in system architecture diagrams.

Task Description
Examine the input diagram for the following layout issues:

1. **Line Crossings:** Check for unreasonable intersections between connection edges or module bounding boxes.
   - *Note: Intentional crossings designed for visual effect are excluded.*
2. **Element Overlaps:** Check for image/element overlaps that negatively impact aesthetics.
   - *Note: Necessary stacking designs (e.g., card stacks) are excluded.*
3. **Text Overflows:** Check if text exceeds its bounding box or overlaps with other images or edges.

Output Format
Output in JSON format wrapped in ```json ```, containing no other text.

Example:
```json{ "layout_issues": [{ "type": "line_crossing", "count": 2, "details": ["Desc 1...", "Desc 2..."] }, { "type": "image_overlap", "count": 1, "details": ["Desc..."] }, { "type": "text_overflow", "count": 1, "details": ["Desc..."] }] } ```

< User >

Please inspect the image for layout anomalies based on the visual content.

Figure 11. The detailed Prompts used in LayoutExamineAgent

Prompt for SystemUnderstandAgent

< System >

Role:
You are a Human Researcher possessing a Master's degree level of cognition. You are responsible for understanding the operating principles of a system based on its architecture diagram and textual description.

Task Description
Comprehend the input system architecture diagram and its accompanying text to provide an explanation of the system's operating principles.

Requirements

- You must simulate **human cognitive levels and thinking patterns** as closely as possible when interpreting the image.

Output Format
Output in JSON format wrapped in ```json, containing no other text.
- Contains a single field: system_understanding (string), representing your understanding of the system's operating principles.

Example:
{ "system_understanding": "..." }```

< User >

Please comprehend the system's operating principles based on the visual content, the paper text, and the graph structure.
Paper Text: **{desc}**
Graph Structure: **{graph}**

Figure 12. The detailed Prompts used in SystemUnderstandAgent
```

Prompt for TextLegibilityAgent

< System >

Role:
You are a Diagram Text Legibility Engineer capable of **meticulously** auditing text within diagrams for clarity issues word-for-word.

Task Description
Read the input system architecture diagram verbatim and inspect it for issues regarding blurriness, incompleteness, and semantic ambiguity.

Requirements

- **Blurriness:** Includes text distortion, non-existent character structures (hallucinations), or text that is mashed into a blur.
- **Incompleteness:** Includes text ending abruptly in inappropriate places, or text artifacts appearing where they should not exist.
- **Semantic Ambiguity:** Includes unintelligible text or text that violates conventional semantics (gibberish).

Output Format
Output in JSON format wrapped in ```json, containing no other text.
- The top-level field is text_legibility_issues, which is a list of dictionaries. Each dictionary contains type, count, and details.
- type values: "Blurry", "Incomplete", "Ambiguous".

Example:
{ "text_legibility_issues": [{ "type": "Blurry", "count": 2, "details": ["Desc 1...", "Desc 2..."] }, { "type": "Incomplete", "count": 1, "details": ["Desc..."] }, { "type": "Ambiguous", "count": 1, "details": ["Desc..."] }] }```

< User >
Please inspect the text in the diagram for issues regarding blurriness, incompleteness, and semantic ambiguity based on the visual content.

Figure 13. The detailed Prompts used in TextLegibilityAgent

S2. Detailed Case Studies

To provide a granular, end-to-end demonstration of our framework, we include a detailed case study in the supplementary archive (`supp.zip`). This case study traces a single research paper, "Browsing Like Human," through both our generation and evaluation pipelines, offering a microscopic view of the system's internal mechanisms.

The study is organized into two directories: `Generation Case` and `Evaluation Case`.

1. Generation Task (`/Generation Case`)

This directory contains all artifacts related to the generation process:

- `Browsing Like Human...pdf`: The original source paper used as input.
- `graph_design_detailed_results.txt`: A comprehensive log detailing the multi-agent reasoning process, from the initial analysis by the *Analyst Agent* to the final refined GraphJSON produced by the collaborative workflow. This file is crucial for understanding the step-by-step construction logic.
- `output figure.png`: The final rendered diagram as a static image.
- `output figure.pptx`: The same diagram in a fully editable PowerPoint format, demonstrating a key output of our system.

2. Evaluation Task (`/Evaluation Case`)

This directory contains the corresponding materials for evaluating the generated output:

- `GroundTruth.png`: The ground-truth diagram extracted from the source paper, serving as the benchmark for comparison.
- `eval_detailed_results.txt`: The full output log from our automated evaluation framework. It shows the step-by-step assessment across all three dimensions (Semantic, Layout, and Visual) and provides detailed justifications for the final scores.

S3. Additional Dataset Statistics and Stability Analysis

In addition to the test set analysis presented in the main paper, this section provides a comprehensive statistical overview of the full training dataset and evaluates the robustness of our automated evaluation framework.

S3.1. Full Dataset Overview

While Sec. 3.3 focused on the curated 108-sample test set, here we present the statistics for the complete **Paper2SysArch-3k** dataset to demonstrate its scale and diversity.

- **Domain Distribution:** As shown in Figure 14(a), the dataset covers four core AI domains. Computer Vision (CV) accounts for the largest share (1066), followed by NLP (873) and Core ML (720), with a significant portion dedicated to AI Systems (341).
- **Conference Source:** Figure 14(b) illustrates that the data originates from **12 top-tier conferences**. This includes premier venues in Vision (CVPR, ICCV, ECCV), Language (ACL, NAACL, EMNLP), Machine Learning (NeurIPS, ICML, ICLR), and Systems (OSDI, NSDI, MLSys), ensuring authoritative and diverse data sources.
- **Temporal Coverage:** As depicted in Figure 14(c), priority is given to recent research, with papers from 2024 (1075) and 2025 (1316) constituting the majority. This ensures the benchmark reflects state-of-the-art architectural trends.

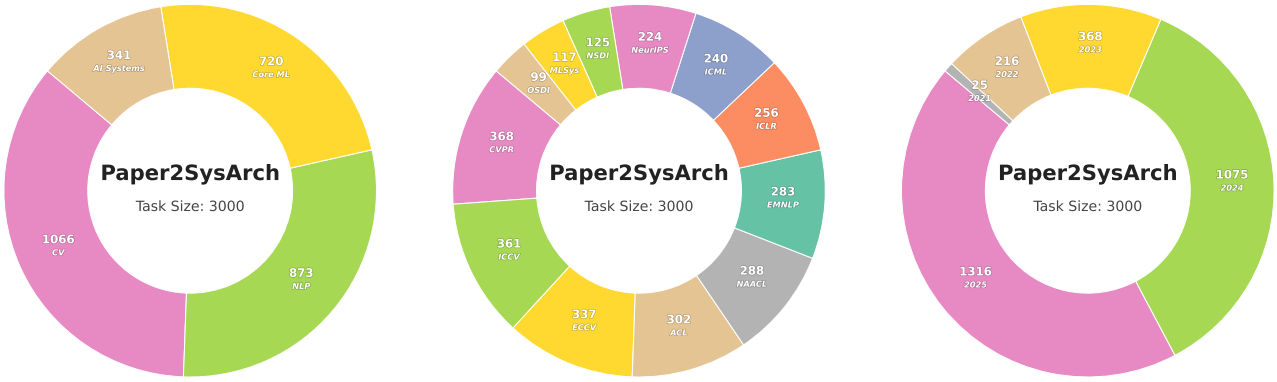
To facilitate a direct inspection of data quality, we provide a **Dataset Sample** in the supplementary archive (`/dataset_sample`), containing representative paper-diagram pairs from each conference.

S3.2. Stability Analysis of Evaluation System

Given the involvement of VLM-based agents in our evaluation framework, the potential for **stochastic fluctuations in scoring is a critical aspect that warrants investigation**. To verify the consistency and reproducibility of the metric, we conducted a repeatability experiment.

Setup: We randomly selected 10 samples (IDs 1-10) from the test set and executed the full evaluation pipeline 5 times for each sample independently, keeping all parameters constant.

Results: As shown in Figure 15, we plot the individual scores (dots), average scores (lines), and min-max ranges (shaded bands) for each ID. The results demonstrate high stability, with most samples showing minimal fluctuation. Even in cases with slight variance (e.g., ID 8), the deviation remains within a narrow, acceptable range. This confirms that the Paper2SysArch evaluation framework produces consistent and reliable assessments.



(a) Research Domains (b) Source Conferences (c) Publication Years

Figure 14. Statistical distribution of the full Paper2SysArch dataset (3,000 samples) across (a) Research Domains, (b) Source Conferences (12 venues), and (c) Publication Years.

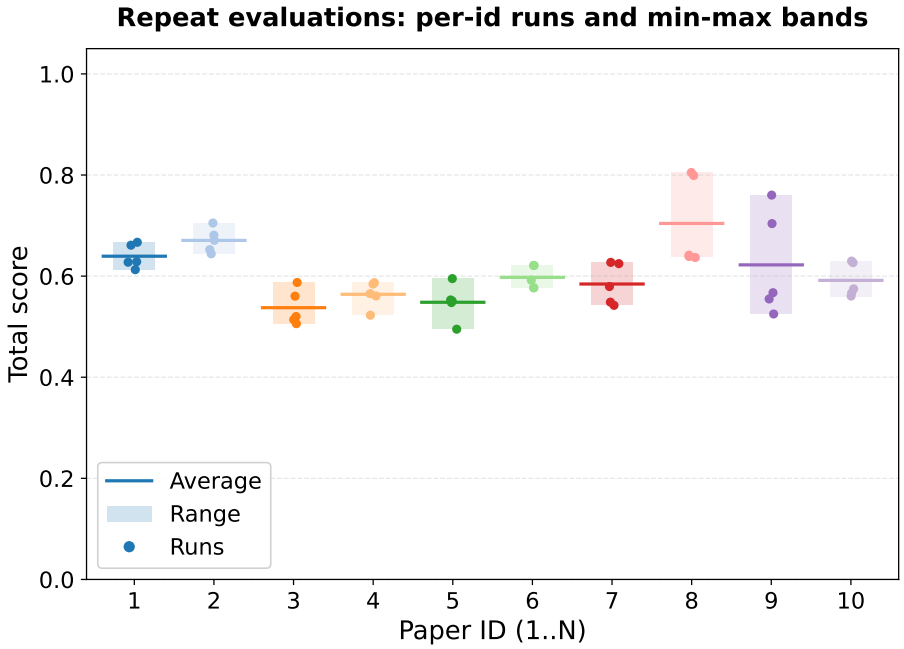


Figure 15. Stability analysis of the evaluation framework. We performed 5 repeat runs for 10 randomly selected paper IDs. The plot shows the individual run scores (dots), the average score (solid line), and the min-max range (shaded area), demonstrating the consistency of our metric.