FORMALIMG: Evaluating Structural Compositional Generalization for T2I Models

Hong-Jie You [1 2] **Jie-Jing Shao** [1] **Xiao-Wen Yang** [1 2] **Zhi-Fan Wu** [3] **Lin-Han Jia** [1] **Lan-Zhe Guo** [1 4]
Yu-Feng Li [1 2]

Abstract

As natural language becomes the primary interface for image generation, evaluating semantic generalization under language instructions is increasingly important. Existing benchmarks emphasize combinations of concepts but rarely examine the internal semantic structure of language. We introduce FORMALIMG, a first-order-logic-based benchmark for structural compositional generalization. Natural language instructions are formalized as logical formulas and we define structural compositional complexity and ε-structural compositional generalizability to measure how model performance changes with increasing semantic dependency. The benchmark includes two evaluation scenarios and 4,000 instructions across multiple complexity levels, assessed through symbolic verification and model-as-judge. Experiments show that mainstream text-to-image models experience clear performance decline as structural complexity grows, with stable performance mainly at low complexity levels. Further analysis indicates that large language models already handle textual structural reasoning well, while the language-to-vision transformation stage forms the significant bottleneck.

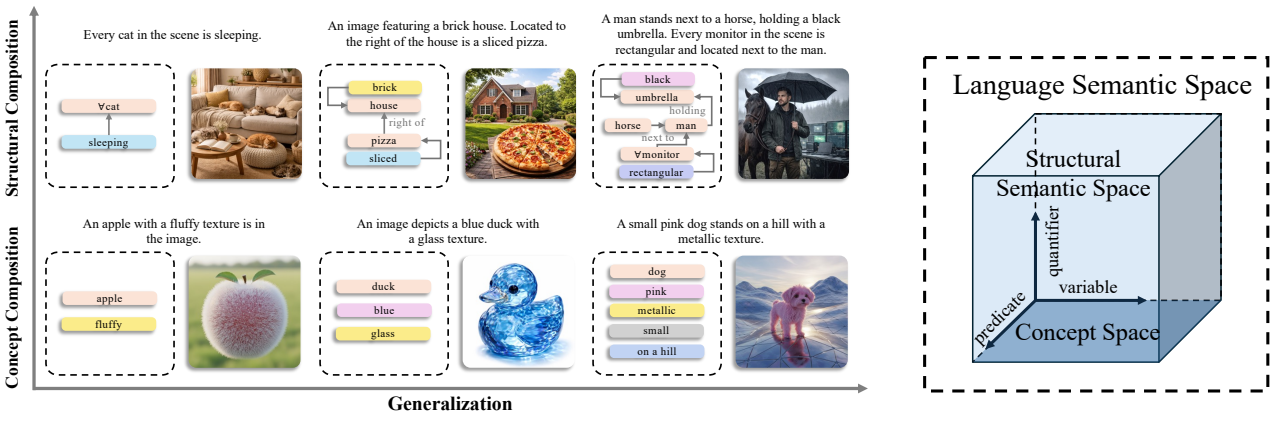
1. Introduction

A long-standing goal of artificial intelligence is to build reliable and general foundation models that assist humans across real-world tasks. Recent advances in large (vision-)language models have accelerated this progress, as natural language instructions now serve as a universal interface for model control. This shift in the control paradigm has profoundly influenced multiple domains. In robotics, control has evolved from manually engineered command sequences (Ajaykumar et al., 2021) to goal-driven natural language instructions (Yao et al., 2025). In programming, assistance has progressed from context-based code completion (Raychev et al., 2014) to conversational code generation (Dong et al., 2025). In recommender systems, preference modeling has shifted from implicit signals such as clicks (Hu et al., 2008) to explicit requirement expression via natural language (Peng et al., 2025). Within this landscape, text-to-image foundation models have emerged as a representative application with a broad user base, with commercial platforms reporting over 22 billion generated images and assets worldwide (Adobe, 2025). These models enable users to transform visual concepts into images using only language instructions, substantially lowering the barrier to content creation. Early models mainly support short or bag-of-words prompts composed of low-level elements such as objects, colors, and textures (Nichol et al., 2022; Rombach et al., 2022; Ramesh et al., 2022). As user demands become more complex, recent models can process full language instructions and capture higher-level semantics, including relationships, attribute binding, and logical structure. As text-to-image systems evolve toward stronger language intelligence, understanding their generalization within a semantic space of combinatorial semantic primitives has become a critical research question.

Early text-to-image benchmarks, such as GenEval (Ghosh et al., 2023) and T2I-CompBench (Huang et al., 2023), evaluate basic linguistic capabilities, including rendering a single object, attribute binding, and simple spatial relations. With the rapid progress of text-to-image systems, some modern models have achieved high overall performance on these benchmarks (Wu et al., 2025; Deng et al., 2025), suggesting that fundamental semantic primitives are largely well mastered. More recent work has begun to examine compositional generalization over these primitives. For example, ConceptMix (Wu et al., 2024) evaluates generalization to unseen combinations of visual concepts by progressively increasing conceptual diversity, such as styles, colors, and object categories. However, ConceptMix mainly

[1]State Key Laboratory of Novel Software Technology, Nanjing University, Nanjing 210023, China [2]School of Artificial Intelligence, Nanjing University, Nanjing 210023, China [3]Alibaba Group [4]School of Intelligence Science and Technology, Nanjing University, Suzhou 215163, China. Correspondence to: Yu-Feng Li <liyf@nju.edu.cn>.

Accepted to the 2nd Workshop on Compositional Learning at ICML 2026, Seoul, South Korea.



(a) (b)

Figure 1. From concept to structural compositional generalization in language-driven text-to-image models. (a) The evolution of generalization in text-to-image models. (b) The structure of the generalization space. The structural semantic space is spanned by variables, predicates, and quantifiers composed through logical connectives, while the concept space can be viewed as a slice within it.

emphasizes the diversity and combination of visual concepts, without systematically characterizing the semantic structure underlying language instructions. In particular, existing benchmarks do not explicitly model how objects, attributes, and relations form a structural semantic space through logical structure, nor do they examine how model performance changes as the dependencies among these elements become more complex.

Motivated by this goal, we introduce FORMALIMG, a benchmark designed to systematically study the structural compositional generalization of text-to-image models within a structural semantic space. The core idea is to explicitly construct a structural semantic space for language instructions and evaluate model performance within this space. To this end, FORMALIMG adopts first-order logic (FOL) as a unified semantic representation, formalizing natural language instructions into logical expressions composed of objects, attributes, unary and binary relations, and quantifiers. The use of FOL provides clear expressive boundaries and strong compositional structure, enabling explicit characterization of the semantic complexity of language instructions and analysis of how model performance changes as semantic complexity increases.

Our main contributions are summarized as follows. **First**, we introduce FORMALIMG, a FOL-based benchmark for studying the semantic generalization of text-to-image models. FORMALIMG contains 4,000 test samples with progressively increasing complexity, providing a solid testbed for systematic analysis of semantic generalization. **Second**, we formally model the semantic structure of language instructions based on FOL, and define structural compositional complexity and ϵ-structural compositional generalizability, thereby providing a computable tool for analyzing the structural compositional generalization of language-driven generative models. **Third**, to balance evaluation rigor and generality, we construct a dual-scenario evaluation protocol consisting of Knolling and Natural settings. The Knolling scenario provides a style-controlled environment, where generated images are symbolized and substituted into logical formulas for formal verification. The Natural scenario serves as a general text-to-image setting, assessing semantic generalization under more realistic and diverse language instructions. **Fourth**, we reveal a hierarchical bottleneck in the structural generalization of current models. Our experiments show that performance consistently degrades as complexity increases. For advanced models, the primary limitation lies in the transformation from language to visual, whereas weaker models exhibit bottlenecks already at the stage of semantic understanding.

2. FORMALIMG

This section introduces the construction of FORMALIMG. Our core idea is to define a structural semantic space grounded in first-order logic, and to construct a set of language instructions whose semantics can be mapped to a single first-order logic formula in this space.

2.1. Structural Semantic Space

Natural language instructions could be formulated as structural expressions generated through recursive composition of vocabulary under grammatical constraints. The vocabulary forms a discrete set of symbols, while grammar provides compositional rules that allow sentences to expand through structural combination. At the semantic level, such recursive composition gives rise to representations of entities, their attributes, and relations between entities, together

with the logical structure among these elements. From a structural perspective, the space of language instructions therefore constitutes a structural semantic space. We formally define the structural semantic space as follows.

Definition 1 (Structural Semantic Space). Let $\mathcal{O}$ denote the set of object symbols, $\mathcal{A}$ the set of attribute symbols, $\mathcal{R}$ the set of relation symbols, and $\mathcal{S}$ the set of semantic operators. Define the set of atomic semantic units as $\mathcal{P} = \{a(o) \mid o \in \mathcal{O},\ a \in \mathcal{A}\} \cup \{r(o_1, o_2) \mid o_1, o_2 \in \mathcal{O},\ r \in \mathcal{R}\}$. The structural semantic space is then defined as $\mathcal{E} = \mathrm{Closure}_{\mathcal{S}}(\mathcal{P})$.

The structural semantic space consists of all well-formed structural expressions that are recursively generated from atomic semantic units under the compositional rules induced by the semantic operators.

The mapping from natural language sentences to structural expressions is generally not unique. Vocabulary choices are open-ended, expressions admit diverse formulations, and quantifier scope as well as coreference resolution may yield multiple interpretative paths. This flexibility makes structural semantic analysis and verification challenging in the full natural language space.

To enable formal analysis of semantic structures in language instructions, we conduct our study in an unambiguous semantic subspace. In this subspace, entities, relations, and scopes are explicitly specified, and semantic composition follows deterministic rules. We adopt FOL as a formal surrogate of this structural semantic space. FOL provides explicit variable binding mechanisms, well-defined quantifier scope, and decidable satisfiability, allowing semantic structures of language instructions to be precisely characterized and supporting subsequent executable verification and complexity analysis.

A first-order formula ϕ consists of variables, predicates, quantifiers, and logical connectives. Variables represent objects; unary predicates represent attributes; binary predicates represent relations; quantifiers specify variable scope; and logical connectives determine the overall logical structure. This yields a compositional and unambiguous semantic representation, enabling explicit analysis of model behavior under different semantic structures. In practice, we instantiate first-order logic in the visual domain via a Domain Specific Language (DSL). The DSL defines a finite vocabulary of objects, attributes, and predicates, providing a closed and controllable platform for our study.

2.2. Structural Compositional Generalization

To investigate model generalization within the structural semantic space, we need to examine model performance across different levels of structural complexity. To this end, we introduce a computable measure of structural compositional complexity. Due to the combinatorial explosion inherent in this space, it is infeasible to enumerate all possible structures. Following prior work (Wu et al., 2024), we therefore conduct analysis along a monotonic and extensible dimension. To capture the reasoning difficulty induced by complex structures, we characterize structural compositional complexity in terms of variable dependency scale.

Definition 2 (Structural Compositional Complexity). Let ϕ be a closed FOL formula in the semantic subspace. According to variable-sharing relations, ϕ is decomposed into connected components with pairwise disjoint variable sets, $\phi = \bigwedge_{i=1}^{N} \phi_i$. Let $\mathrm{Var}(\phi_i)$ denote the set of quantified variables appearing in component ϕ_i. The structural compositional complexity of ϕ is defined as

$$K(\phi) = \max_i |\mathrm{Var}(\phi_i)|. \tag{1}$$

This measure captures the maximal dependency scale among variables in the formula. When $K(\phi) = 1$, substructures associated with different variables are independent. When $K(\phi) > 1$, variables are connected through binary predicates, and the model must perform multi-entity joint reasoning.

During dataset construction, we further constrain each instruction to contain only a single connected component, so that $K(\phi)$ faithfully reflects the depth of coupled relational reasoning.

Definition 3 (ε-Structural Compositional Generalizability). Let Φ denote the set of all FOL instruction formulas. For any integer $k \in \mathbb{N}$, define the complexity level $\Phi_k = \{\phi \in \Phi \mid K(\phi) = k\}$. Let x_ϕ denote the natural language instruction corresponding to formula ϕ. Let $S(y, \phi) \in \{0, 1\}$ be a semantic satisfaction function indicating whether a generated output y satisfies ϕ. For a model f, let $SR_k(f) = \mathbb{E}_{\phi \sim \mathcal{U}(\Phi_k), y \sim f(\cdot|x_\phi)}[S(y, \phi)]$ denote the average satisfaction rate of f at complexity level k. Given a performance threshold $\varepsilon \in (0, 1)$, the ε-structural compositional generalizability of f is defined as

$$G_\varepsilon(f) = \max\{K \mid SR_k(f) \geq \varepsilon,\ \forall k \leq K\}. \tag{2}$$

Structural compositional complexity characterizes the scale of joint reasoning required by a formula. However, it does not by itself describe model behavior. To study model performance in the structured semantic space, we examine how accuracy varies with structural complexity. An ideal compositional reasoning model should maintain stable performance as structural complexity increases. The ε-structural compositional generalizability measures the range of structural compositional complexity levels over which a model

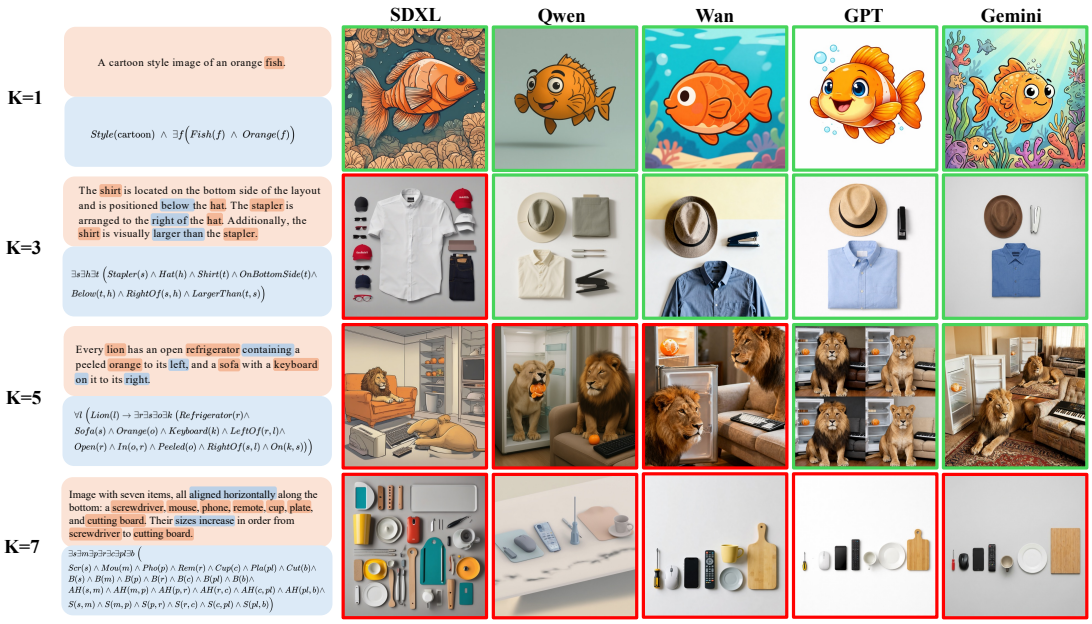


Figure 2. Generation examples of models under instructions with increasing complexity. Green borders denote images that satisfy the instruction, while red denote failures.

can stably maintain acceptable performance. The condition $SR_k(f) \geq \varepsilon$ for all $k \leq K$ requires that the model achieve a satisfaction rate above the threshold at every complexity level up to K. Consequently, $G_\varepsilon(f)$ represents the largest continuous structural compositional complexity level that the model can reliably generalize.

2.3. Benchmark Construction

Inspired by (Wu et al., 2024), we generate data through a controlled LLM-based synthesis pipeline with both syntax and semantic checking.

Instead of directly sampling logical forms, we first sample a generation configuration in DSL domain that defines the target properties of each instruction, such as the desired K, object categories, quantifier structure and predicates. This configuration serves as a structural prior of an instruction.

Conditioned on this configuration, an LLM jointly generates a natural language instruction and its corresponding DSL expression. Generating both representations within a single semantic context encourages globally coherent descriptions and avoids the unnatural or contradictory structures that often arise from programmatic enumeration of logical forms.

The generated data are then subjected to formal verification implemented in Python. The DSL is checked for syntactic validity, variable scope consistency, domain compliance, and correct complexity assignment. An additional LLM-based semantic check detects potential semantic conflicts and ensures alignment between the natural language instructions and their corresponding DSL representations. Samples failing any verification step are discarded and regenerated.

To ensure both experimental rigor and broader applicability, we construct two complementary scenarios: Knolling and Natural. The Knolling scenario features structured object arrangements with visual grounding, facilitating symbolic abstraction and formal verification. The Natural scenario reflects general text-to-image generation, allowing us to examine structural compositional generalization under more open semantic conditions. Examples are shown in Figure 2.

2.4. Evaluation

We adopt scenario-specific evaluation strategies. The Knolling setting is evaluated via formal symbolic verification, whereas the Natural setting relies on model-as-judge for assessment.

Knolling. Let y denote a generated image and ϕ the target DSL instruction. A symbolic scene representation $S(y)$ is constructed through a hybrid parsing process: object categories and attributes are recognized using Qwen3-VL-8B (Bai et al., 2025a), while geometric properties such as positions, sizes, and spatial relations are computed from detected object layouts. The correctness of the generation is evaluated by logical satisfaction: $\text{Score}(y, \phi) = 1$ if $S(y) \models \phi$, and 0 otherwise.

Natural. For natural image evaluation, we employ Gemini-3-Pro (Google DeepMind, 2025) as an automated

judge. The model takes the generated image together with the natural language instruction and the corresponding DSL instruction, and produces a binary decision indicating whether the instruction is satisfied.

3. Experiments and Key Observations

3.1. Experimental Setup

Benchmark Composition. FORMALIMG consists of two evaluation scenarios: Knolling and Natural. Each scenario contains 2,000 test samples, covering complexity levels from $K = 1$ to $K = 10$. Each complexity level includes 200 samples. We further group complexity levels into three subsets: Easy ($K = 1$ to 3), Medium ($K = 4$ to 6), and Hard ($K = 7$ to 10).

Model Selection. We evaluate multiple text-to-image models, including both advanced close-source models (e.g., GPT-Image-1.5 (OpenAI, 2025b) and Gemini-3-Pro-Image (Google, 2025b)) and well-validated open-source models (e.g., SDXL (Podell et al., 2024) and Qwen-Image (Wu et al., 2025)).

Metric. Following the evaluation method described in Section 2.4, we compute the average instruction satisfaction rate. Results are reported separately for each complexity level K, along with the overall average success rate. Under the threshold $\varepsilon = 0.7$, we compute and report the ε-structural compositional generalizability $G_{0.7}$ for each model.

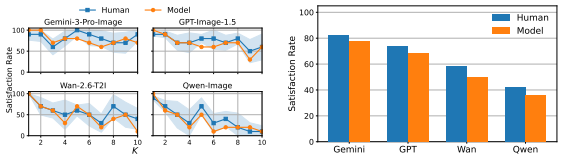
3.2. Limited Structural Compositional Generalization

Table 1 shows that in both the Natural and Knolling evaluation scenarios, the instruction satisfaction rate consistently decreases as K increases.

As complexity grows, a clear performance gap gradually emerges between close-source and open-source models. Although some open-source models remain competitive at low K, their performance declines rapidly as K increases. In the high-K regime, several open-source models exhibit near-zero satisfaction rates. In contrast, close-source models demonstrate a more gradual degradation trend and maintain non-trivial performance across a broader range of complexity levels.

Within the close-source model group, performance differences also become more pronounced as K increases. At low K, models perform comparably; however, the gap widens significantly in the high-K regime. Gemini-3-Pro-Image maintains a higher instruction satisfaction rate and exhibits a relatively smoother decline, yet its performance still decreases steadily as complexity increases.

From the perspective of structural compositional generalizability $G_{0.7}$, most models achieve stable generalization only at low structural complexity levels. Among close-source models, the majority can reliably generalize only up to $K = 2$. Even the strongest model, Gemini-3-Pro-Image, generalizes only up to $K = 6$ in the Natural scenario and $K = 4$ in the Knolling scenario. For open-source models, none satisfies the generalization threshold at $K = 2$. Some models fail to generalize even in single-object generation tasks involving global constraints and multiple attribute bindings.



(a) Satisfaction rates across K. *(b)* Ranking consistency across models.

Figure 3. Comparison between model-based judgments and human evaluation.

These findings indicate that current text-to-image models still lack robust structural compositional generalization when handling language instructions, highlighting a substantial gap that remains to be addressed.

3.3. Human Evaluation and Reliability of Model-as-Judge

To assess the reliability of automatic evaluation in the Natural scenario, we conduct a human evaluation on a stratified subset of the benchmark, following the scale adopted in prior work (Wu et al., 2024). We randomly sample 100 instructions (10 per complexity level $K \in [1, 10]$) and collect generated images from four text-to-image models, yielding 400 image–instruction pairs. Forty volunteers participate in the evaluation, each rating 50 samples, resulting in 2,000 ratings in total. Each image is evaluated by five annotators on a five-point Likert scale for semantic compliance, and the final score is the average rating. a sample is considered to satisfy the language instruction if its mean score is at least 4.

Human ratings show high internal consistency (Cronbach's $\alpha = 0.882$). Comparing binarized human judgments with the model-as-judge outputs yields an accuracy of 0.805 and an F1 score of 0.840. Ranking consistency is measured by the Spearman correlation between human and automatic model rankings across K, yielding a mean correlation of 0.844. Figure 3 shows that automatic satisfaction rates closely follow human trends across K. In most cases, the automatic results fall within the variability caused by inter-annotator disagreement. Overall model rankings from automatic evaluation exactly match those from human evalua-

Table 1. Instruction satisfaction rates of text-to-image models on FORMALIMG across two evaluation scenarios and K. E, M and H denote the average performance on Easy, Medium and Hard subset. O denotes the overall average.

Model	K										E	M	H	O	$G_{0.7}$
	1	2	3	4	5	6	7	8	9	10					
Natural															
Gemini-3-Pro-Image (Google, 2025b)	92.5	78.5	**76.0**	**79.5**	**78.5**	**74.0**	**68.0**	**76.0**	**71.5**	**65.5**	82.3	**77.3**	**70.2**	**76.0**	**6**
Gemini-2.5-Flash-Image (Google, 2025a)	90.0	63.5	54.0	54.5	61.5	49.0	38.0	39.5	42.0	28.5	69.2	55.0	37.0	52.1	1
GPT-Image-1 (OpenAI, 2025a)	95.0	81.0	71.0	76.5	57.0	59.0	61.5	54.5	51.0	47.5	82.3	64.2	53.6	65.4	4
GPT-Image-1.5 (OpenAI, 2025b)	**96.0**	**84.0**	73.5	73.5	63.5	63.5	62.0	59.5	52.5	53.0	**84.5**	66.8	56.8	68.1	4
Seedream-4.5 (Chen et al., 2025)	95.5	83.5	55.0	67.0	58.0	48.5	41.5	44.5	35.5	30.0	78.0	57.8	37.9	55.9	2
Wan-2.6-T2I (Wang et al., 2025)	92.5	72.0	58.5	54.0	45.0	45.0	36.5	41.0	30.0	34.0	74.3	48.0	35.4	50.9	2
SD1.5 (Rombach et al., 2022)	55.5	10.0	2.0	0.5	0.0	0.0	0.0	0.0	0.0	0.0	22.5	0.2	0.0	6.8	0
SDXL (Podell et al., 2024)	72.5	26.5	7.5	2.0	1.0	0.5	0.0	0.0	0.0	0.0	35.5	1.2	0.0	11.0	1
Flux1.dev (Black Forest Labs, 2024)	73.0	47.5	29.5	23.5	18.5	13.5	10.5	9.0	4.0	1.5	50.0	18.5	6.2	23.1	1
Qwen-Image (Wu et al., 2025)	90.0	66.5	43.0	38.0	35.5	26.5	17.0	22.0	13.0	11.0	66.5	33.3	15.8	36.3	1
Hunyuan-3.0 (Cao et al., 2026)	77.5	58.5	40.5	35.0	32.0	23.5	15.0	18.0	14.0	9.5	58.8	30.2	14.1	32.4	1
BAGEL (Deng et al., 2025)	85.0	49.0	30.0	22.0	26.5	14.5	6.5	9.0	7.0	1.5	54.7	21.0	6.0	25.1	1
Knolling															
Gemini-3-Pro-Image (Google, 2025b)	93.0	**90.0**	**80.0**	**70.0**	**66.5**	**63.5**	**58.5**	**49.0**	**53.0**	**47.0**	**87.7**	**66.7**	**51.9**	**67.1**	**4**
Gemini-2.5-Flash-Image (Google, 2025a)	**94.5**	73.0	57.5	53.5	36.5	35.0	23.5	18.5	18.5	12.5	75.0	41.7	18.2	42.3	2
GPT-Image-1 (OpenAI, 2025a)	94.0	71.0	59.5	49.5	45.5	37.5	36.5	24.0	20.5	16.0	74.8	44.2	24.2	45.4	2
GPT-Image-1.5 (OpenAI, 2025b)	92.0	75.5	62.5	52.0	51.5	48.0	42.0	30.0	28.0	20.0	76.7	50.5	30.0	50.2	2
Seedream-4.5 (Chen et al., 2025)	85.0	65.0	48.5	34.0	33.5	26.5	21.0	17.0	12.0	12.5	66.2	31.3	15.6	35.5	1
Wan-2.6-T2I (Wang et al., 2025)	63.0	53.0	39.5	30.0	30.0	28.0	24.0	14.5	18.0	12.0	51.8	29.3	17.1	31.2	0
SD1.5 (Rombach et al., 2022)	43.5	17.0	4.0	0.5	0.0	0.0	0.0	0.0	0.0	0.0	21.5	0.2	0.0	6.5	0
SDXL (Podell et al., 2024)	50.0	31.5	7.5	1.0	0.5	0.0	0.0	0.0	0.0	0.0	29.7	0.5	0.0	9.1	0
Flux1.dev (Black Forest Labs, 2024)	56.5	35.0	16.0	9.0	2.5	3.5	1.5	0.0	0.5	0.0	35.8	5.0	0.5	12.5	0
Qwen-Image (Wu et al., 2025)	65.0	39.0	27.0	23.0	13.0	7.5	8.0	4.0	4.5	1.5	43.7	14.5	4.5	19.3	0
Hunyuan-3.0 (Cao et al., 2026)	49.5	35.0	20.0	12.5	5.0	3.5	3.5	3.0	3.5	0.0	34.8	7.0	2.5	13.6	0
BAGEL (Deng et al., 2025)	38.0	32.0	5.0	1.5	1.0	0.0	0.5	0.0	0.0	0.0	25.0	0.8	0.1	7.8	0

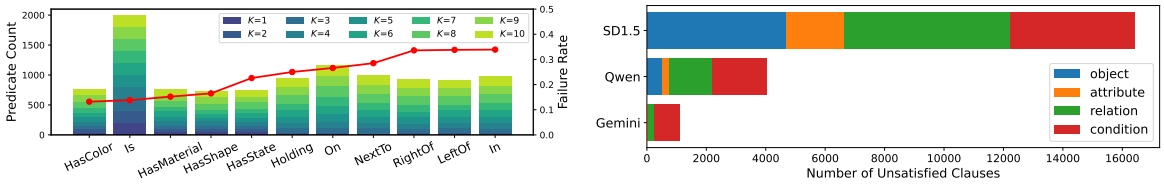
tion, although automatic scores are slightly more conservative.

These findings demonstrate that semantic compliance in FORMALIMG can be consistently assessed by human evaluators, and that the model-as-judge framework provides a reliable and scalable alternative to human evaluation.

3.4. Clause-Level Analysis of Compositional Failures

To further understand the causes of failure in structural compositional generalization for text-to-image models, we conduct fine-grained statistical analysis in the Natural scenario, which contains richer vocabulary of objects, attributes, and relations. We first convert each DSL expression into conjunctive normal form, representing the semantic specification as a conjunction of multiple clauses. During the model-as-judge evaluation process, the judge model is required to return the list of clauses it considers satisfied. We then categorize these clauses into four semantic groups according to their functional roles: object, attribute, relation, and condition. Based on this categorization, we compute the frequency of major predicates across different K levels, the failure rates associated with each predicate, and the proportion of failures across clause categories for different models. The results are shown in Figure 4.

Figure 4a shows that the distribution of predicates across K is overall relatively balanced. Apart from the `Is` predicate, which appears in all instructions, the remaining predicates also occur at comparable frequencies. In terms of failure rates, models perform relatively stably on unary predicates that describe objects and attributes, whereas binary predicates that express relations exhibit noticeably higher failure rates. Figure 4b further illustrates the differences in failure patterns across models. For the weaker model SD1.5, fail-



(a) Predicates count and failure rate.

(b) Unsatisfied count across models.

Figure 4. Failure analysis across predicates and clause types.

ures occur frequently in both object and relation clauses, indicating limited capability in handling basic semantic atoms. For the stronger model Gemini-3-Pro-Image, the failure proportions for object, attribute, and relation clauses decrease substantially, with errors primarily concentrated in condition clauses. The mid-tier model Qwen-Image exhibits behavior between these two extremes. These observations suggest that current advanced text-to-image models have become relatively reliable in attribute-level semantic control, yet still face clear bottlenecks when handling complex structural compositions involving multiple interacting constraints.

3.5. Modality-Level Diagnosis

A key question is whether the performance degradation arises from insufficient semantic understanding or from failures during language-to-vision generation. To localize the bottleneck, we evaluate LLMs in a pure text setting under the Knolling scenario: given a natural language instruction, the model outputs a symbolic layout specification, which is verified using the same DSL interpreter.

As shown in Figure 5 and Table 2, modern language models maintain high satisfaction rates across all complexity levels. Gemini-3-Pro-Thinking and DeepSeek-V3.2-Thinking achieve overall scores of 97.0% and 97.6%, respectively, while sustaining over 90% accuracy even at high K, with both reaching $G_{0.7} = 10$. Reasoning variants consistently outperform their non-reasoning counterparts, particularly at higher complexity. For example, Qwen3-30B-A3B improves from $G_{0.7} = 4$ to $G_{0.7} = 10$ when reasoning is enabled. These results indicate that structural compositional generalization is largely preserved in the language domain.

In contrast, the best text-to-image model achieves only 67.1% overall performance and degrades rapidly with increasing K. To identify the primary source of error, we conduct controlled comparisons on Gemini-3-Pro-Image and Qwen-Image under both text-only and full generation settings. For Gemini-3-Pro-Image we use its text-output mode, while for Qwen-Image we evaluate its language backbone Qwen2.5-VL-7B (Bai et al., 2025b). As shown in Figure 6, Gemini-3-Pro-Image maintains nearly 90% performance in text mode, whereas its image generation performance declines sharply with complexity, and the gap between the two modes widens as K increases. This indicates that instruction understanding remains robust while the main limitation arises during language-to-vision generation. In contrast, Qwen-Image exhibits a large gap between text and image outputs even at low complexity. As K increases, its text performance deteriorates rapidly, narrowing the gap and suggesting that language understanding becomes the dominant bottleneck at high complexity.

Overall, these findings reveal a hierarchical bottleneck in structural compositional generalization: stronger models are primarily limited by language-to-visual generation, whereas weaker models are constrained by both semantic understanding and visual generation.

3.6. Intermediate Layout Representations

The analysis in Section 3.5 shows that the language-to-vision transformation stage constitutes a significant bottleneck. Providing an explicit instance as an intermediate reference may reduce instantiation ambiguity and stabilize the generation process. We conduct experiments in the Knolling scenario using intermediate layout representations (both textual and visual) generated by DeepSeek-V3.2-Thinking (97.6% text-only satisfaction rate) as conditions for image generation. Full results are provided in the supplementary material.

Intermediate layouts stabilize the generation process and improve structural compositional generalization. When the original instruction is preserved, incorporating intermediate layouts yields performance gains at nearly all complexity levels for stronger models, with more pronounced improvements under the visual layout condition. For Gemini-3-Pro-Image, both layout representations increase structural compositional generalizability to $G_{0.7} = 7$.

However, the original instruction remains crucial for conveying logical semantic constraints. The layout corresponds to a specific satisfying instance rather than the underlying logical constraints themselves. When the original instruction is removed, textual layouts lead to substantial performance drops, while visual layouts maintain performance closer to the original setting. Comparing the two formats, visual layouts produce more stable and larger overall gains, likely be-

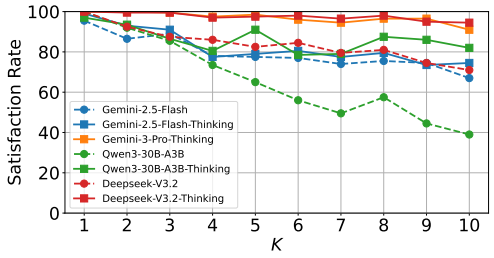


Figure 5. Performance of LLMs on the Knolling text task across K.

Table 2. Performance summary of LLMs on the Knolling text task.

Model	E	M	H	O	$G_{0.7}$
Gemini-2.5-Flash (Google, 2025a)	90.3	77.5	72.8	79.5	9
Gemini-2.5-Flash-Thinking (Google, 2025a)	94.3	79.0	76.2	82.5	**10**
Gemini-3-Pro-Thinking (Google DeepMind, 2025)	**99.8**	97.3	94.6	97.0	**10**
Qwen3-30B-A3B (Yang et al., 2025)	92.5	64.8	47.6	66.3	4
Qwen3-30B-A3B-Thinking (Yang et al., 2025)	92.3	83.3	83.6	86.2	**10**
Deepseek-V3.2 (Liu et al., 2025)	93.3	84.3	76.5	83.9	**10**
Deepseek-V3.2-Thinking (Liu et al., 2025)	99.7	**97.5**	**96.0**	**97.6**	**10**

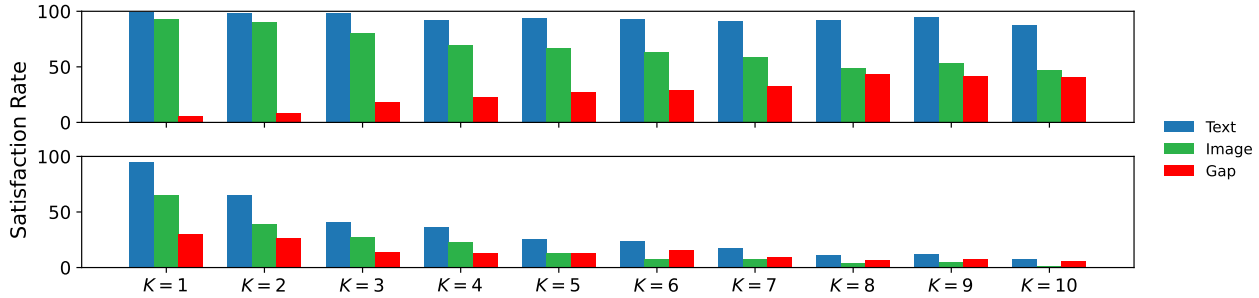


Figure 6. Comparison of text and image setting across K. The upper plot shows **Gemini-3-Pro-Image**, and the lower plot shows **Qwen-Image**.

cause current models receive limited exposure to coordinate-based textual representations during training. Overall, intermediate representations can partially mitigate failures in language-to-vision generation, though their effectiveness depends on alignment between the representation format and the model's conditioning modality.

4. Related Work

Language-Driven Text-to-Image Models. Early text-to-image approaches were based on GANs (Goodfellow et al., 2014; Mirza & Osindero, 2014) or CVAEs (Kingma & Welling, 2014; Sohn et al., 2015). With the emergence of diffusion models (Ho et al., 2020) and multimodal encoders such as CLIP (Radford et al., 2021), modern systems including GLIDE (Nichol et al., 2022), Stable Diffusion (Rombach et al., 2022), and DALL·E 2 (Ramesh et al., 2022) began to leverage text embeddings for conditional generation. More recently, LLMs and VLMs have led to more powerful text encoders (Saharia et al., 2022; Black Forest Labs, 2024; Wu et al., 2025) and unified architectures (Lu et al., 2023; Xie et al., 2025; Deng et al., 2025), making natural language the primary interface for visual generation.

Evaluation of Text-to-Image Models. Early evaluation relied on image quality metrics such as FID (Heusel et al., 2017), IS (Salimans et al., 2016), and text–image similarity (Hessel et al., 2021). Dedicated benchmarks then introduced fine-grained evaluation of basic semantic capabilities (Huang et al., 2023; 2025; Ghosh et al., 2023). However, such evaluations usually focus on isolated elements or fixed templates. More recent studies investigate compositional generalization (Kamath et al., 2025; Li et al., 2024; Wu et al., 2024). For example, ConceptMix (Wu et al., 2024) evaluates generalization to unseen combinations by controlling the number of visual concepts. Despite this progress, existing benchmarks mainly emphasize specific skills or novel visual combinations, while paying limited attention to the linguistic structure of instructions.

5. Conclusion

This paper introduces FORMALIMG, an FOL-based benchmark for structural compositional generalization, and defines structural compositional complexity together with ε-structural compositional generalizability to characterize the stable performance of text-to-image models under varying scales of semantic dependency. Experimental results show that existing models exhibit performance degradation as structural complexity increases, with stable generalization remaining confined to low-complexity regimes. Further analysis reveals a hierarchical bottleneck phenomenon, and shows that incorporating intermediate layout representations can partially mitigate performance degradation.

References

Adobe. Adobe revolutionizes ai-assisted creativity with firefly, the all-in-one home for ai content creation, with new partner and firefly models. https://news.adobe.com/news/2025/04/adobe-revolutionizes-ai-assisted-creativity-firefly, 2025.

Ajaykumar, G., Steele, M., and Huang, C. A survey on end-user robot programming. *ACM Computing Surveys*, 54(8):164:1–164:36, 2021.

Bai, S., Cai, Y., Chen, R., Chen, K., Chen, X., Cheng, Z., Deng, L., Ding, W., Gao, C., Ge, C., et al. Qwen3-vl technical report. *arXiv preprint arXiv:2511.21631*, 2025a.

Bai, S., Chen, K., Liu, X., Wang, J., Ge, W., Song, S., Dang, K., Wang, P., Wang, S., Tang, J., et al. Qwen2.5-vl technical report. *arXiv preprint arXiv:2502.13923*, 2025b.

Black Forest Labs. FLUX. https://github.com/black-forest-labs/flux, 2024.

Cao, S., Chen, H., Chen, P., Cheng, Y., Cui, Y., Deng, X., Dong, Y., Gong, K., Gu, T., Gu, X., et al. Hunyuanimage 3.0 technical report. *arXiv preprint arXiv:2509.23951*, 2026.

Chen, Y., Gao, Y., Gong, L., Guo, M., Guo, Q., Guo, Z., Hou, X., Huang, W., Huang, Y., Jian, X., et al. Seedream 4.0: Toward next-generation multimodal image generation. *arXiv preprint arXiv:2509.20427*, 2025.

Deng, C., Zhu, D., Li, K., Gou, C., Li, F., Wang, Z., Zhong, S., Yu, W., Nie, X., Song, Z., Guang, S., and Fan, H. Emerging properties in unified multimodal pretraining. *arXiv preprint arXiv:2505.14683*, 2025.

Dong, Y., Jiang, X., Qian, J., Wang, T., Zhang, K., Jin, Z., and Li, G. A survey on code generation with llm-based agents. *arXiv preprint arXiv:2508.00083*, 2025.

Ghosh, D., Hajishirzi, H., and Schmidt, L. GenEval: An object-focused framework for evaluating text-to-image alignment. In *Advances in Neural Information Processing Systems*, volume 36, pp. 52132–52152, 2023.

Goodfellow, I. J., Pouget-Abadie, J., Mirza, M., Xu, B., Warde-Farley, D., Ozair, S., Courville, A. C., and Bengio, Y. Generative adversarial nets. In *Advances in Neural Information Processing Systems*, volume 27, pp. 2672–2680, 2014.

Google. Gemini 2.5 flash and gemini 2.5 flash image model card. https://storage.googleapis.com/deepmind-media/Model-Cards/Gemini-2-5-Flash-Model-Card.pdf, 2025a.

Google. Gemini 3 pro image model card. https://storage.googleapis.com/deepmind-media/Model-Cards/Gemini-3-Pro-Image-Model-Card.pdf, 2025b.

Google DeepMind. Gemini 3 pro model card. https://storage.googleapis.com/deepmind-media/Model-Cards/Gemini-3-Pro-Model-Card.pdf, 2025.

Hessel, J., Holtzman, A., Forbes, M., Bras, R. L., and Choi, Y. CLIPScore: A reference-free evaluation metric for image captioning. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing*, pp. 7514–7528, 2021.

Heusel, M., Ramsauer, H., Unterthiner, T., Nessler, B., and Hochreiter, S. GANs trained by a two time-scale update rule converge to a local nash equilibrium. In *Advances in Neural Information Processing Systems*, volume 30, pp. 6626–6637, 2017.

Ho, J., Jain, A., and Abbeel, P. Denoising diffusion probabilistic models. In *Advances in Neural Information Processing Systems*, volume 33, pp. 6840–6851, 2020.

Hu, Y., Koren, Y., and Volinsky, C. Collaborative filtering for implicit feedback datasets. In *Proceedings of the IEEE International Conference on Data Mining*, pp. 263–272, 2008.

Huang, K., Sun, K., Xie, E., Li, Z., and Liu, X. T2I-CompBench: A comprehensive benchmark for open-world compositional text-to-image generation. In *Advances in Neural Information Processing Systems*, volume 36, pp. 78723–78747, 2023.

Huang, K., Duan, C., Sun, K., Xie, E., Li, Z., and Liu, X. T2I-CompBench++: An enhanced and comprehensive benchmark for compositional text-to-image generation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 47(5):3563–3579, 2025.

Kamath, A., Chang, K., Krishna, R., Zettlemoyer, L., Hu, Y., and Ghazvininejad, M. GenEval 2: Addressing benchmark drift in text-to-image evaluation. *arXiv preprint arXiv:2512.16853*, 2025.

Kingma, D. P. and Welling, M. Auto-encoding variational bayes. In *International Conference on Learning Representations*, 2014.

Li, B., Lin, Z., Pathak, D., Li, J., Fei, Y., Wu, K., Ling, T., Xia, X., Zhang, P., Neubig, G., and Ramanan, D. GenAI-Bench: Evaluating and improving compositional text-to-visual generation. *arXiv preprint arXiv:2406.13743*, 2024.

Liu, A., Mei, A., Lin, B., Xue, B., Wang, B., Xu, B., Wu, B., Zhang, B., Lin, C., Dong, C., et al. Deepseek-v3.2: Pushing the frontier of open large language models. *arXiv preprint arXiv:2512.02556*, 2025.

Lu, J., Clark, C., Zellers, R., Mottaghi, R., and Kembhavi, A. UNIFIED-IO: A unified model for vision, language, and multi-modal tasks. In *International Conference on Learning Representations*, 2023.

Mirza, M. and Osindero, S. Conditional generative adversarial nets. *arXiv preprint arXiv:1411.1784*, 2014.

Nichol, A. Q., Dhariwal, P., Ramesh, A., Shyam, P., Mishkin, P., McGrew, B., Sutskever, I., and Chen, M. GLIDE: towards photorealistic image generation and editing with text-guided diffusion models. In *Proceedings of the International Conference on Machine Learning*, pp. 16784–16804, 2022.

OpenAI. Gpt image 1 model. https://platform.openai.com/docs/models/gpt-image-1, 2025a.

OpenAI. New chatgpt images is here. https://openai.com/index/new-chatgpt-images-is-here/, 2025b.

Peng, Q., Liu, H., Huang, H., Yang, J., Yang, Q., and Shao, M. A survey on llm-powered agents for recommender systems. In *Findings of the Association for Computational Linguistics: EMNLP 2025*, pp. 11574–11583, 2025.

Podell, D., English, Z., Lacey, K., Blattmann, A., Dockhorn, T., Müller, J., Penna, J., and Rombach, R. SDXL: improving latent diffusion models for high-resolution image synthesis. In *International Conference on Learning Representations*, 2024.

Radford, A., Kim, J. W., Hallacy, C., Ramesh, A., Goh, G., Agarwal, S., Sastry, G., Askell, A., Mishkin, P., Clark, J., Krueger, G., and Sutskever, I. Learning transferable visual models from natural language supervision. In *Proceedings of the International Conference on Machine Learning*, volume 139, pp. 8748–8763, 2021.

Ramesh, A., Dhariwal, P., Nichol, A., Chu, C., and Chen, M. Hierarchical text-conditional image generation with CLIP latents. *arXiv preprint arXiv:2204.06125*, 2022.

Raychev, V., Vechev, M. T., and Yahav, E. Code completion with statistical language models. In *Proceedings of the ACM SIGPLAN Conference on Programming Language Design and Implementation*, pp. 419–428, 2014.

Rombach, R., Blattmann, A., Lorenz, D., Esser, P., and Ommer, B. High-resolution image synthesis with latent diffusion models. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 10674–10685, 2022.

Saharia, C., Chan, W., Saxena, S., Li, L., Whang, J., Denton, E. L., Ghasemipour, S. K. S., Lopes, R. G., Ayan, B. K., Salimans, T., Ho, J., Fleet, D. J., and Norouzi, M. Photorealistic text-to-image diffusion models with deep language understanding. In *Advances in Neural Information Processing Systems*, volume 35, pp. 36479–36494, 2022.

Salimans, T., Goodfellow, I. J., Zaremba, W., Cheung, V., Radford, A., and Chen, X. Improved techniques for training GANs. In *Advances in Neural Information Processing Systems*, volume 29, pp. 2226–2234, 2016.

Sohn, K., Lee, H., and Yan, X. Learning structured output representation using deep conditional generative models. In *Advances in Neural Information Processing Systems*, volume 28, pp. 3483–3491, 2015.

Wang, A., Ai, B., Wen, B., Mao, C., Xie, C.-W., Chen, D., Yu, F., Zhao, H., Yang, J., Zeng, J., et al. Wan: Open and advanced large-scale video generative models. *arXiv preprint arXiv:2503.20314*, 2025.

Wu, C., Li, J., Zhou, J., Lin, J., Gao, K., Yan, K., ming Yin, S., Bai, S., Xu, X., Chen, Y., et al. Qwen-image technical report. *arXiv preprint arXiv:2508.02324*, 2025.

Wu, X., Yu, D., Huang, Y., Russakovsky, O., and Arora, S. ConceptMix: A compositional image generation benchmark with controllable difficulty. In *Advances in Neural Information Processing Systems*, volume 37, pp. 86004–86047, 2024.

Xie, J., Mao, W., Bai, Z., Zhang, D. J., Wang, W., Lin, K. Q., Gu, Y., Chen, Z., Yang, Z., and Shou, M. Z. Show-o: One single transformer to unify multimodal understanding and generation. In *International Conference on Learning Representations*, 2025.

Yang, A., Li, A., Yang, B., Zhang, B., Hui, B., Zheng, B., Yu, B., Gao, C., Huang, C., Lv, C., et al. Qwen3 technical report. *arXiv preprint arXiv:2505.09388*, 2025.

Yao, X., Zhou, H., Mees, O., Meng, Y., Xiao, T., Bisk, Y., Oh, J., Johns, E., Shridhar, M., Shah, D., Thomason, J., Huang, K., Chai, J., Bing, Z., and Knoll, A. Bridging language and action: A survey of language-conditioned robot manipulation. *arXiv preprint arXiv:2312.10807*, 2025.

A. Benchmark Details

A.1. Vocabulary

The benchmark is constructed from a predefined vocabulary that specifies the set of object categories, attribute values, scene properties, predicates, and logical operators. This controlled vocabulary ensures consistent expression generation while allowing sufficient compositional diversity.

Natural Scenario. The vocabulary for the natural scenarios includes:

Natural Vocabulary

Object Classes: person, man, woman, child, cat, dog, bird, horse, fish, elephant, lion, butterfly, table, chair, sofa, bed, bookshelf, desk, lamp, wardrobe, rug, mirror, plant, clock, book, cup, laptop, plate, bottle, key, fork, knife, spoon, bowl, pan, refrigerator, apple, banana, orange, cake, pizza, sandwich, car, bicycle, bus, boat, tree, flower, rock, house, bench, basket, fruit, hat, shirt, shoes, jacket, glasses, backpack, phone, keyboard, monitor, headphones, hammer, screwdriver, umbrella, ball, toy car, teddy bear.
Colors: red, green, blue, yellow, black, white, brown, orange, pink, purple, gray.
Materials: wooden, metal, glass, plastic, fluffy, fabric, ceramic, leather, paper, stone, brick.
Shapes: round, square, cylindrical, rectangular, triangular, oval
States: open, closed, on, off, standing, sitting, lying, kneeling, running, jumping, flying, swimming, sleeping, smiling, crying, broken, rusty, dirty, wet, burning, melting, sliced, peeled, bitten, empty, full, blooming, withered.
Scene Properties: photo, oil painting, cartoon, cyberpunk, pixel art.
Unary Predicates: Is, HasColor, HasMaterial, HasShape, HasState.
Binary Predicates: LeftOf, RightOf, On, In, Holding, NextTo.
Scene Predicates: IsStyle.
Logical Operators: and, or, implies, exists, forall, not.

Knolling Scenario. The vocabulary for the knolling scenarios includes:

Knolling Vocabulary

Object Classes: laptop, mouse, keyboard, monitor, phone, headphones, camera, game_controller, remote, book, pen, scissors, stapler, ruler, calculator, glasses, watch, wallet, keys, lighter, backpack, handbag, suitcase, shoe, hat, shirt, jeans, glove, sock, tie, hammer, screwdriver, wrench, pliers, drill, bottle, cup, fork, knife, spoon, plate, bowl, pan, cutting_board, apple, lemon, donut, sandwich, pizza, carrot.
Colors: red, green, blue, yellow, black, white, brown, pink, gray.
Unary Predicates: Is, Has, OnLeftSide, OnRightSide, OnTopSide, OnBottomSide, InCenter.
Binary Predicates: LeftOf, RightOf, Above, Below, AlignedHorizontally, AlignedVertically, LargerThan, SmallerThan.
Logical Operators: and, or, implies, exists, forall, not.

Together, this vocabulary defines the domain from which DSL programs are generated for the benchmark.

A.2. DSL Grammar

Formal expressions in our benchmark are expressed in a DSL based on FOL. The grammar of the DSL is defined using a BNF-style notation:

BNF Grammar of the DSL

$$
\begin{aligned}
\langle expr \rangle &::= \langle predicate \rangle \\
&\mid (and\ \langle expr \rangle\ \langle expr \rangle^{+}) \\
&\mid (or\ \langle expr \rangle\ \langle expr \rangle^{+}) \\
&\mid (implies\ \langle expr \rangle\ \langle expr \rangle) \\
&\mid (not\ \langle expr \rangle) \\
&\mid (exists\ \langle variable \rangle\ \langle expr \rangle) \\
&\mid (forall\ \langle variable \rangle\ \langle expr \rangle) \\
\langle predicate \rangle &::= \langle unary\_predicate \rangle \\
&\mid \langle binary\_predicate \rangle \\
&\mid \langle scene\_predicate \rangle \\
\langle unary\_predicate \rangle &::= (p\ \langle variable \rangle\ \langle value \rangle) \\
&\mid (p\ \langle variable \rangle) \\
\langle binary\_predicate \rangle &::= (p\ \langle variable \rangle\ \langle variable \rangle) \\
\langle scene\_predicate \rangle &::= (p\ \langle value \rangle)
\end{aligned}
$$

Here p denotes a predicate symbol defined in the vocabulary. During the syntax checking stage, we implement a Python-based parser that validates each generated DSL expression against the above grammar rules to ensure syntactic correctness.

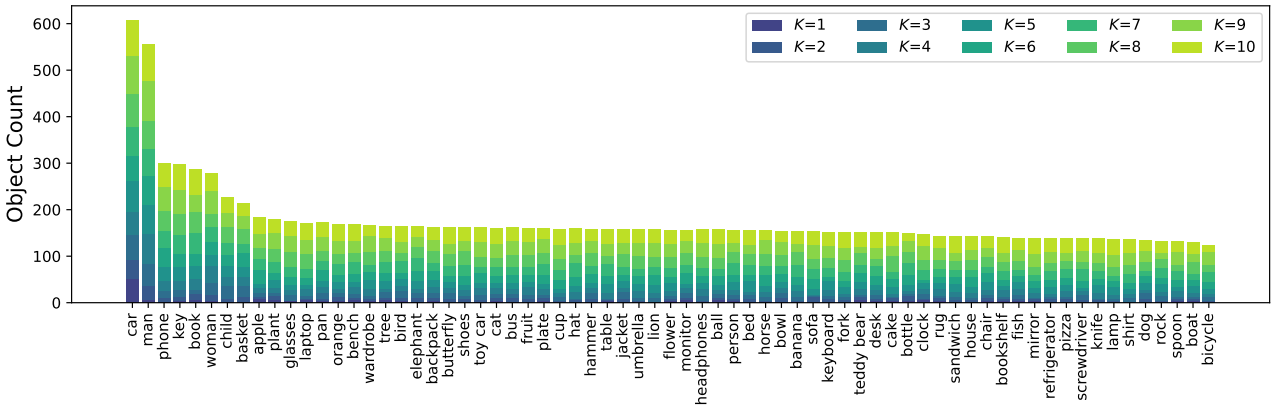


Figure 7. Object distribution in the natural scenario.

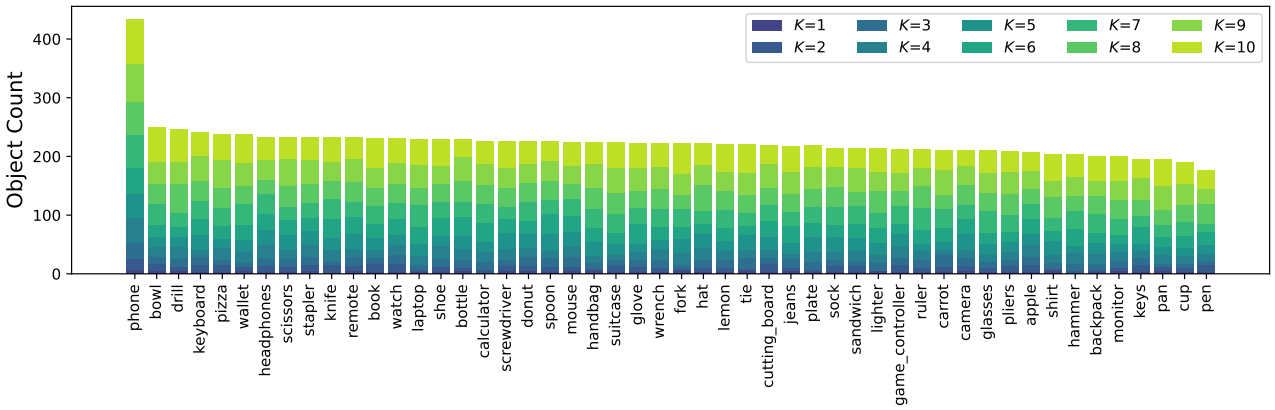
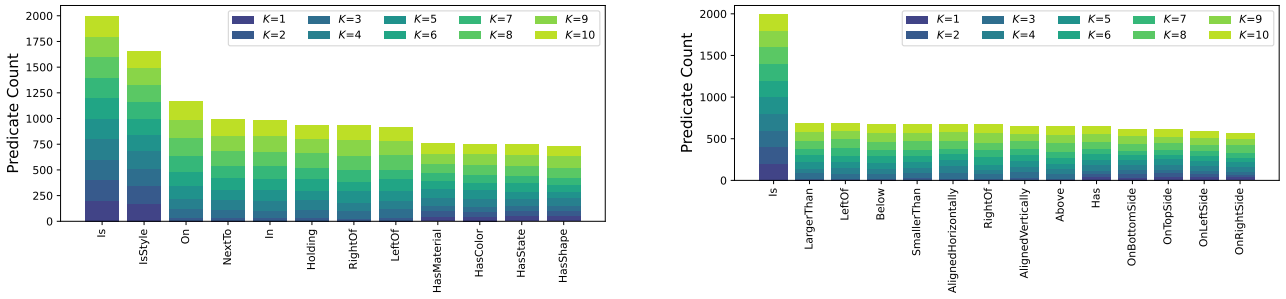


Figure 8. Object distribution in the knolling scenario.



(a) Natural scenario

(b) Knolling scenario

Figure 9. Predicate distribution in the benchmark across the two scenarios.

A.3. Object and Predicate Distribution

We report the distributions of object categories and predicates for both the natural and knolling scenarios. During data generation, predicates are sampled approximately uniformly from the predefined domain, except for several structural predicates (e.g., Is, IsStyle) that naturally occur more frequently when describing object attributes. The resulting statistics show that most vocabulary items are relatively balanced, and the distributions across different values of K are also reasonably uniform.

After generation, all samples undergo a validation stage to ensure syntax and semantic correctness. This step may slightly adjust the final frequency of some vocabulary items.

A.4. Predicate Evaluation Rules in Knolling Scenario

In the knolling scenario, predicates are evaluated based on the 2D bounding boxes. The center of the object is defined as the center of its bounding box, and the area is computed from the bounding box size. The following rules define the evaluation of all predicates used in this scenario.

- **OnLeftSide**(o) holds if the horizontal coordinate of the object center is smaller than half of the image width.
- **OnRightSide**(o) holds if the horizontal coordinate of the object center is greater than half of the image width.
- **OnTopSide**(o) holds if the vertical coordinate of the object center is smaller than half of the image height.
- **OnBottomSide**(o) holds if the vertical coordinate of the object center is greater than half of the image height.
- **LeftOf**(a, b) holds if the right boundary of object a is located to the left of the horizontal center of object b.
- **RightOf**(a, b) holds if the left boundary of object a is located to the right of the horizontal center of object b.
- **Above**(a, b) holds if the bottom boundary of object a is above the vertical center of object b.
- **Below**(a, b) holds if the top boundary of object a is below the vertical center of object b.
- **AlignedHorizontally**(a, b) holds if the difference between the vertical coordinates of the object centers is smaller than a fixed threshold.
- **AlignedVertically**(a, b) holds if the difference between the horizontal coordinates of the object centers is smaller than a fixed threshold.
- **LargerThan**(a, b) holds if the area of the bounding box of object a is larger than that of object b.
- **SmallerThan**(a, b) holds if the area of the bounding box of object a is smaller than that of object b.

These rules provide a deterministic procedure for verifying whether a predicted arrangement satisfies the logical constraints in the knolling scenario.

Table 3. Effect of layout representation and query setting across K. Text and image layouts are compared under both *With Query* and *Without Query* conditions.

Layout	Query	Model	K										O	$G_{0.7}$
			1	2	3	4	5	6	7	8	9	10		
Text	With	Gemini-3-Pro-Image (Google, 2025b)	91.0↓2.0	89.5↓0.5	83.5↑3.5	76.5↑6.5	71.5↑5.0	70.0↑6.5	70.0↑11.5	62.5↑13.5	59.5↑6.5	51.0↑4.0	72.5↑5.4	7↑3
		GPT-Image-1.5 (OpenAI, 2025b)	95.5↑3.5	82.0↑6.5	67.5↑5.0	55.0↑3.0	58.5↑7.0	48.5↑0.5	39.0↓3.0	35.5↑5.5	33.0↑5.0	21.5↑1.5	53.6↑3.4	2(0)
		Seedream-4.5 (Chen et al., 2025)	77.5↓7.5	70.0↑5.0	51.0↑2.5	38.0↑4.0	33.0↓0.5	22.5↓4.0	16.5↓4.5	11.0↓6.0	9.5↓2.5	4.0↓8.5	33.3↓2.2	2↑1
	Without	Gemini-3-Pro-Image (Google, 2025b)	64.0↓29.0	83.5↓6.5	74.0↓6.0	61.5↓8.5	61.5↓5.0	54.5↓9.0	46.5↓12.0	43.5↓5.5	42.0↓11.0	38.5↓8.5	57.0↓10.1	0↓4
		GPT-Image-1.5 (OpenAI, 2025b)	58.5↓33.5	78.5↑3.0	64.5↑2.0	52.0(0.0)	41.5↓10.0	37.0↓11.0	26.5↓15.5	21.0↓9.0	17.5↓10.5	12.5↓7.5	41.0↓9.2	0↓2
		Seedream-4.5 (Chen et al., 2025)	43.5↓41.5	40.0↓25.0	10.5↓38.0	5.0↓29.0	6.0↓27.5	8.0↓18.5	6.5↓14.5	1.0↓16.0	7.5↓4.5	1.5↓11.0	13.0↓22.5	0↓1
Image	With	Gemini-3-Pro-Image (Google, 2025b)	94.0↑1.0	93.5↑3.5	88.0↑8.0	86.0↑16.0	78.0↑11.5	71.5↑8.0	71.5↑13.0	65.0↑16.0	68.5↑15.5	59.0↑12.0	77.5↑10.4	7↑3
		GPT-Image-1.5 (OpenAI, 2025b)	95.5↑3.5	84.0↑8.5	74.0↑11.5	65.0↑13.0	59.0↑7.5	57.5↑9.5	45.5↑3.5	36.0↑6.0	33.5↑5.5	23.5↑3.5	57.4↑7.2	3↑1
		Seedream-4.5 (Chen et al., 2025)	85.5↑0.5	74.5↑9.5	61.5↑13.0	49.5↑15.5	42.5↑9.0	39.0↑12.5	42.5↑21.5	25.5↑8.5	28.5↑16.5	22.5↑10.0	47.2↑11.7	2↑1
	Without	Gemini-3-Pro-Image (Google, 2025b)	66.5↓26.5	87.0↓3.0	79.0↓1.0	78.0↑8.0	70.5↑4.0	66.0↑2.5	68.0↑9.5	59.0↑10.0	55.5↑2.5	52.5↑5.5	68.2↑1.1	0↓4
		GPT-Image-1.5 (OpenAI, 2025b)	64.0↓28.0	81.0↑5.5	61.5↓1.0	53.0↑1.0	47.5↓4.0	47.0↓1.0	37.0↓5.0	30.0(0.0)	26.5↓1.5	22.0↑2.0	47.0↓3.2	0↓2
		Seedream-4.5 (Chen et al., 2025)	51.0↓34.0	69.0↑4.0	41.5↓7.0	38.0↑4.0	37.0↑3.5	35.0↑8.5	24.5↑3.5	25.0↑8.0	24.0↑12.0	23.5↑11.0	36.9↑1.4	0↓1

Table 4. Ablation on prompt length and object count.

Scenario	Model	$K \in \{1, 2\}$	$K = 10$
Knolling	Gemini-3-Pro-Image	76.5	47.0
Knolling	GPT-Image-1.5	65.5	20.0
Natural	Gemini-3-Pro-Image	72.5	65.5
Natural	GPT-Image-1.5	85.0	53.0

B. Additional Experimental Results

B.1. Ablation on Prompt Length and Object Count

To disentangle structural complexity from prompt length and object count, we construct a controlled subset with $K \in \{1, 2\}$ and 10 objects, and compare it directly with the original $K = 10$ setting.

As shown in Table 4, this controlled setting is consistently easier than the original $K = 10$ benchmark. This indicates that the degradation at high K cannot be explained by prompt length or object count alone, and that structural complexity forms a distinct challenge for compositional generalization.

B.2. Intermediate Layout Representations

This section provides the detailed results underlying the layout analysis in Section 3.5. We use DeepSeek-V3.2-Thinking outputs from the text-only setting (97.6% satisfaction) as approximate layout plans, compare textual and visual layouts, and test both with and without the original instruction.

Table 3 shows that layouts are most helpful when the original instruction is preserved, and that visual layouts are consistently more effective than textual coordinate layouts. When the instruction is removed, performance drops substantially for all models, indicating that layouts serve mainly as auxiliary structural guidance rather than a replacement for the instruction.

B.3. Evaluation Reliability

This section provides the detailed evidence behind the reliability discussion in the main text. For the Natural scenario, we evaluate 100 sampled instructions and 400 image–instruction pairs with five annotators per sample, yielding 2,000 ratings in total.

The main evidence is that human ratings are internally consistent (Cronbach's $\alpha = 0.882$), model-as-judge outputs agree well with binarized human judgments (accuracy 0.805, F1 0.840), and model rankings are preserved across K (mean Spearman correlation 0.844). Figure 3 visualizes these trends.

For the Knolling scenario, we manually refine annotations on 100 sampled generations and compare them against the automatic parser. The parser reaches 96.67% precision, 97.80% recall, and 97.23% F1 under IoU@0.9, and color prediction accuracy is 95.43% on matched objects, indicating limited symbolic evaluation noise.

Table 5. Cross-judge validation in the Natural scenario using GPT-5.5 as the judge model.

Rank	Model	Accuracy
1	Gemini-3-Pro-Image	81.40%
2	GPT-Image-1.5	78.60%
3	Wan-2.6-T2I	61.35%
4	Qwen-Image	47.95%

We also report a cross-judge check with GPT-5.5 in Table 5. Although absolute scores are slightly higher, the model ranking is unchanged, supporting the robustness of the comparative conclusions to judge choice.

C. Evaluation Examples

C.1. Natural Scenario Evaluation Examples

We present an example evaluation case in the natural scenario. The prompt, the corresponding DSL representation in CNF-style form, the input image, and the response produced by the VLM are shown below.

In this representation, the top level may contain a conjunction that groups multiple dependency closures. Each closure is internally converted into a conjunction of clauses following a CNF structure.

Example 1

Prompt: In this image, all directions are relative to the frame. A rock is inside a wet jacket. Every fish is on the jacket to the left of the rock.

DSL in CNF-Style: (exists ?j (exists ?r (forall ?f (and (Is ?j 'jacket') (HasState ?j 'wet') (In ?r ?j) (or (not (Is ?f 'fish')) (On ?f ?j)) (Is ?r 'rock') (or (not (Is ?f 'fish')) (LeftOf ?f ?r))))))

VLM Output:

```
{
    "satisfies_constraint": true,
    "satisfied_clauses": [
        "(Is ?j 'jacket')",
        "(HasState ?j 'wet')",
        "(In ?r ?j)",
```

```
        "(or (not (Is ?f 'fish')) (On ?f ?j))",
        "(Is ?r 'rock')",
        "(or (not (Is ?f 'fish')) (LeftOf ?f ?r))"
    ]
}
```

Example 2

Prompt: In this image, all directions are relative to the frame. This is an oil painting style scene featuring a blooming tree. A rectangular bookshelf is located next to the tree. A teddy bear sits on the bookshelf, and a butterfly is on the teddy bear. To the right of the tree is a bus, and to the right of the bus is a person holding a hammer.

DSL in CNF-Style: (and (IsStyle 'oil painting') (exists ?t (exists ?bs (exists ?tb (exists ?bf (exists ?b (exists ?p (exists ?h (and (Is ?t 'tree') (HasState ?t 'blooming') (NextTo ?bs ?t) (RightOf ?b ?t) (HasShape ?bs 'rectangular') (On ?tb ?bs) (Is ?b 'bus') (RightOf ?p ?b) (Is ?bs 'bookshelf') (Is ?tb 'teddy bear') (On ?bf ?tb) (Is ?p 'person') (Holding ?p ?h) (Is ?bf 'butterfly') (Is ?h 'hammer'))))))))))

VLM Output:

```
{
    "satisfies_constraint": true,
    "satisfied_clauses": [
        "(IsStyle 'oil painting')",
        "(Is ?t 'tree')",
        "(HasState ?t 'blooming')",
        "(NextTo ?bs ?t)",
        "(RightOf ?b ?t)",
        "(HasShape ?bs 'rectangular')",
        "(On ?tb ?bs)",
        "(Is ?b 'bus')",
        "(RightOf ?p ?b)",
        "(Is ?bs 'bookshelf')",
        "(Is ?tb 'teddy bear')",
        "(On ?bf ?tb)",
        "(Is ?p 'person')",
        "(Holding ?p ?h)",
        "(Is ?bf 'butterfly')",
        "(Is ?h 'hammer')"
```

```
    ]
}
```

Example 3

Prompt: In this image, all directions are relative to the frame. A photo where every child holds a round pizza, with a toy car to their right.

DSL in CNF-Style: (and (IsStyle 'photo') (forall ?c (exists ?p (exists ?t (and (or (not (Is ?c 'child')) (Is ?p 'pizza')) (or (not (Is ?c 'child')) (HasShape ?p 'round')) (or (not (Is ?c 'child')) (Holding ?c ?p)) (or (not (Is ?c 'child')) (Is ?t 'toy car')) (or (not (Is ?c 'child')) (RightOf ?t ?c))))))

VLM Output:

```
{
    "satisfies_constraint": false,
    "satisfied_clauses": [
        "(IsStyle 'photo')",
        "(or (not (Is ?c 'child')) (Is ?p 'pizza'))",
        "(or (not (Is ?c 'child')) (HasShape ?p 'round'))",
        "(or (not (Is ?c 'child')) (Holding ?c ?p))",
        "(or (not (Is ?c 'child')) (Is ?t 'toy car'))"
    ]
}
```

Example 4

Prompt: In this image, all directions are relative to the frame. This is a cyberpunk style scene. A table is situated to the left of a bookshelf. All fruits in the image are round and contained within the bookshelf. A child stands to the left of the table, holding a green hat with a ball inside. A laptop is positioned to the right of the child.

DSL in CNF-Style: (and (IsStyle 'cyberpunk') (exists ?bs (exists ?t (forall ?f (exists ?c (exists ?h (exists ?b (exists ?l (and (Is ?bs 'bookshelf') (LeftOf ?t ?bs) (or (not (Is ?f 'fruit')) (In ?f ?bs)) (Is ?t 'table') (LeftOf ?c ?t) (or (not (Is ?f 'fruit')) (HasShape ?f 'round')) (Is ?c 'child') (Holding ?c ?h) (RightOf ?l ?c) (HasColor ?h 'green') (In ?b ?h) (Is ?l 'laptop') (Is ?h 'hat') (Is ?b 'ball'))))))))))

VLM Output:

```
{
    "satisfies_constraint": false,
    "satisfied_clauses": [
        "(IsStyle 'cyberpunk')",
        "(Is ?bs 'bookshelf')",
        "(or (not (Is ?f 'fruit')) (In ?f ?bs))",
        "(Is ?t 'table')",
        "(LeftOf ?c ?t)",
        "(or (not (Is ?f 'fruit')) (HasShape ?f 'round'))",
        "(Is ?c 'child')",
        "(Holding ?c ?h)",
        "(RightOf ?l ?c)",
        "(HasColor ?h 'green')",
        "(In ?b ?h)",
        "(Is ?l 'laptop')",
        "(Is ?h 'hat')",
        "(Is ?b 'ball')"
    ]
}
```

C.2. Knolling Scenario Evaluation Examples

We present an example evaluation case in the knolling scenario. The prompt, the corresponding DSL representation, the grounded image, and the search tree of DSL interpreter.

Example 5

Prompt: Generate a Knolling-style image featuring a neatly arranged collection. A handbag is positioned on the left side of the scene. To the right of the handbag is a keyboard. Below the handbag is a calculator. A cutting board is located to the right of the calculator and is visibly larger than it. A book is arranged below the keyboard. Additionally, every apple present in the scene is red and is smaller than the calculator.

DSL: (exists ?handbag (exists ?keyboard (exists ?calculator (exists ?cutting_board (exists ?book (and (Is ?handbag 'handbag') (OnLeftSide ?handbag) (Is ?keyboard 'keyboard') (RightOf ?keyboard ?handbag) (Is ?calculator 'calculator') (Below ?calculator ?handbag) (Is ?cutting_board 'cutting_board') (RightOf ?cutting_board ?calculator)

(LargerThan ?cutting_board ?calculator) (Is ?book 'book') (Below ?book ?keyboard) (forall ?apple (implies (Is ?apple 'apple') (and (Has ?apple 'red') (SmallerThan ?apple ?calculator))))))))))

Search Tree:

```
[OK] Exists ?handbag satisfied by Obj #9
|
\- [OK] AND satisfied
   |
   |- [OK] Object is 'handbag'
   |
   |- [OK] Object is on left side
   |
   |- [OK] Exists ?calculator satisfied by Obj #10
   |  |
   |  \- [OK] AND satisfied
   |     |
   |     |- [OK] Object is 'calculator'
   |     |- [OK] calculator is Below handbag
   |     |- [OK] Forall ?apple satisfied (Triggered 6 times)
   |     |
   |     \- [OK] Exists ?cutting_board satisfied by Obj #8
   |        |
   |        \- [OK] AND satisfied
   |           |
   |           |- [OK] Object is 'cutting_board'
   |           |- [OK] cutting_board is RightOf calculator
   |           \- [OK] cutting_board is LargerThan calculator
   |
   \- [OK] Exists ?keyboard satisfied by Obj #7
      |
      \- [OK] AND satisfied
         |
         |- [OK] Object is 'keyboard'
         |- [OK] keyboard is RightOf handbag
         |
         \- [OK] Exists ?book satisfied by Obj #0
            |
            \- [OK] AND satisfied
               |
               |- [OK] Object is 'book'
```

```
                \- [OK] book is Below keyboard
```

Example 6

Prompt: Generate a Knolling-style image. A calculator is positioned on the bottom side of the scene. For every watch present, there is a screwdriver aligned horizontally with it, and the watch appears visually smaller than the calculator.

DSL: (exists ?c (and (Is ?c 'calculator') (OnBottomSide ?c) (forall ?w (implies (Is ?w 'watch') (exists ?s (and (Is ?s 'screwdriver') (AlignedHorizontally ?s ?w) (SmallerThan ?w ?c)))))))

Search Tree:

```
[FAIL] Exists ?c failed
|
|- [FAIL] Candidate #0 failed
|   |
|   \- [FAIL] AND check failed
|      |
|      |- [OK] Object is 'calculator'
|      |- [OK] Success
|      \- [FAIL] Forall failed at Obj #1
|         |
|         \- [FAIL] IMPLIES fails (antecedent true, consequent false)
|            |
|            |- [OK] Object is 'watch'
|            \- [FAIL] AND check failed
|               |
|               |- [OK] Success
|               \- [FAIL] Exists ?s failed
|                  |
|                  |- [FAIL] Candidate #0 failed
|                  |   |
|                  |   \- [FAIL] AND check failed
|                  |      |
|                  |      \- [FAIL] Is -> Type mismatch: Expected 'screwdriver', got
↪  'calculator'
|                  |
|                  |- [FAIL] Candidate #1 failed
```

```
|                  |  |
|                  |  \- [FAIL] AND check failed
|                  |     |
|                  |     \- [FAIL] Is -> Type mismatch: Expected 'screwdriver', got
↪  'watch'
|                  |
|                  |- [FAIL] Candidate #2 failed
|                  |  |
|                  |  \- [FAIL] AND check failed
|                  |     |
|                  |     \- [FAIL] Is -> Type mismatch: Expected 'screwdriver', got
↪  'watch'
|                  |
|                  |- [FAIL] Candidate #3 failed
|                  |  |
|                  |  \- [FAIL] AND check failed
|                  |     |
|                  |     |- [OK] Object is 'screwdriver'
|                  |     \- [FAIL] AlignedHorizontally -> Not AlignedHorizontally
↪  (diff=139.0)
|                  |
|                  \- [FAIL] Candidate #4 failed
|                     |
|                     \- [FAIL] AND check failed
|                        |
|                        |- [OK] Object is 'screwdriver'
|                        \- [FAIL] AlignedHorizontally -> Not AlignedHorizontally
↪  (diff=139.0)
|
|- [FAIL] Candidate #1 failed
|  |
|  \- [FAIL] AND check failed
|     |
|     \- [FAIL] Is -> Type mismatch: Expected 'calculator', got 'watch'
|
|- [FAIL] Candidate #2 failed
|  |
|  \- [FAIL] AND check failed
|     |
|     \- [FAIL] Is -> Type mismatch: Expected 'calculator', got 'watch'
|
|- [FAIL] Candidate #3 failed
|  |
|  \- [FAIL] AND check failed
|     |
|     \- [FAIL] Is -> Type mismatch: Expected 'calculator', got 'screwdriver'
|
\- [FAIL] Candidate #4 failed
   |
   \- [FAIL] AND check failed
      |
      \- [FAIL] Is -> Type mismatch: Expected 'calculator', got 'screwdriver'
```

D. Examples of Each Failure Category

To analyze the failure modes of the model, we categorize errors based on the unsatisfied clauses in the CNF form of the DSL constraints.

If the clause is an `Is` predicate specifying the existence or identity of an object, the failure is categorized as an **Object Failure**. If the clause is a unary predicate describing an attribute of an object, such as `HasState` or `HasShape`, the failure is categorized as an **Attribute Failure**. If the clause is a binary predicate describing a relation between two objects, it is categorized as a **Relation Failure**. Finally, if the clause takes the CNF implication form `(or (not A) B)`, which corresponds to a conditional constraint in the original DSL specification, the failure is categorized as a **Condition Failure**.

D.1. Object Failure

Example 7

Prompt: In this image, all directions are relative to the frame. A photo of a woman wearing a red hat, with every butterfly in the scene resting on it. She is holding a teddy bear. To her right is a black monitor with a round cup containing a fork placed on it.

DSL in CNF-Style: (and (IsStyle 'photo') (exists ?w (exists ?h (forall ?b (exists ?tb (exists ?m (exists ?c (exists ?f (and (Is ?w 'woman') (On ?h ?w) (Holding ?w ?tb) (HasColor ?h 'red') (or (not (Is ?b 'butterfly')) (On ?b ?h)) (RightOf ?m ?w) (Is ?tb 'teddy bear') (Is ?h 'hat') (HasColor ?m 'black') (On ?c ?m) (Is ?m 'monitor') (HasShape ?c 'round') (In ?f ?c) (Is ?c 'cup') (Is ?f 'fork'))))))))))

VLM Output:

```
{
    "satisfies_constraint": false,
    "satisfied_clauses": [
        "(IsStyle 'photo')",
        "(Is ?w 'woman')",
        "(On ?h ?w)",
        "(Holding ?w ?tb)",
        "(HasColor ?h 'red')",
        "(or (not (Is ?b 'butterfly')) (On ?b ?h))",
        "(RightOf ?m ?w)",
        "(Is ?tb 'teddy bear')",
        "(Is ?h 'hat')",
        "(HasColor ?m 'black')",
        "(On ?c ?m)",
        "(HasShape ?c 'round')",
        "(In ?f ?c)",
        "(Is ?c 'cup')",
        "(Is ?f 'fork')"
    ]
}
```

Reason: Missing the monitor object.

D.2. Attribute Failure

Example 8

Prompt: In this image, all directions are relative to the frame. A child stands to the left of a tree, holding a round plate with a sliced fish on it.

DSL in CNF-Style: (exists ?c (exists ?t (exists ?p (exists ?f (and (Is ?c 'child') (HasState ?c 'standing') (LeftOf ?c ?t) (Holding ?c ?p) (Is ?t 'tree') (Is ?p 'plate') (HasShape ?p 'round') (On ?f ?p) (Is ?f 'fish') (HasState ?f 'sliced')))))

VLM Output:

```
{
    "satisfies_constraint": false,
    "satisfied_clauses": [
        "(Is ?c 'child')",
        "(HasState ?c 'standing')",
        "(LeftOf ?c ?t)",
        "(Holding ?c ?p)",
        "(Is ?t 'tree')",
        "(Is ?p 'plate')",
        "(HasShape ?p 'round')",
        "(On ?f ?p)",
        "(Is ?f 'fish')"
    ]
}
```

Reason: The fish is not sliced.

D.3. Relation Failure

Example 9

Prompt: In this image, all directions are relative to the frame. A cyberpunk style scene shows a woman standing next to a chair, with a sleeping dog beside it. All butterflies are located to the left of the woman. To the right of the dog sits a toy car containing an apple.

DSL in CNF-Style: (and (IsStyle 'cyberpunk') (exists ?w (exists ?c (exists ?d (forall ?b (exists ?t (exists ?a (and (Is

?w ’woman’) (HasState ?w ’standing’) (NextTo ?w ?c) (or (not (Is ?b ’butterfly’)) (LeftOf ?b ?w)) (Is ?c ’chair’) (NextTo ?d ?c) (Is ?d ’dog’) (HasState ?d ’sleeping’) (RightOf ?t ?d) (Is ?t ’toy car’) (In ?a ?t) (Is ?a ’apple’)))))))))

VLM Output:

```
{
    "satisfies_constraint": false,
    "satisfied_clauses": [
        "(IsStyle 'cyberpunk')",
        "(Is ?w 'woman')",
        "(HasState ?w 'standing')",
        "(NextTo ?w ?c)",
        "(or (not (Is ?b 'butterfly')) (LeftOf ?b ?w))",
        "(Is ?c 'chair')",
        "(NextTo ?d ?c)",
        "(Is ?d 'dog')",
        "(HasState ?d 'sleeping')",
        "(RightOf ?t ?d)",
        "(Is ?t 'toy car')",
        "(Is ?a 'apple')"
    ]
}
```

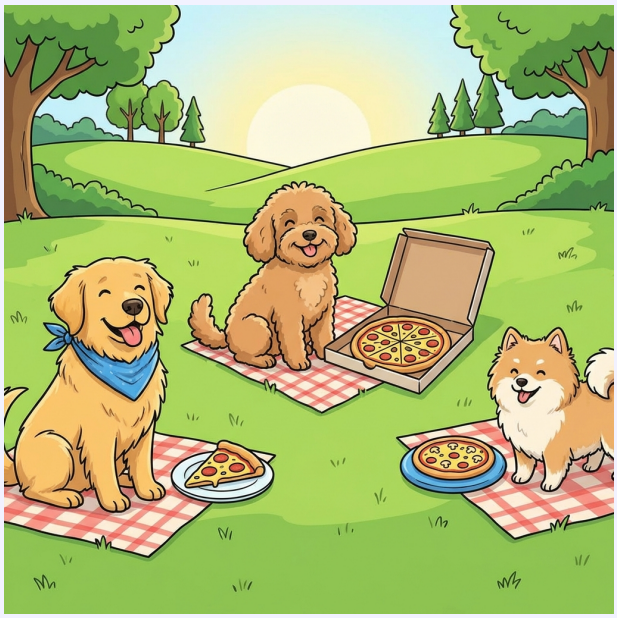
Reason: The apple is not in the toy car.

D.4. Condition Failure

Example 10

Prompt: In this image, all directions are relative to the frame. A cartoon style scene where every fluffy dog has a pizza to its right.

DSL in CNF-Style: (and (IsStyle ’cartoon’) (forall ?d (exists ?p (and (or (not (Is ?d ’dog’)) (not (HasMaterial ?d ’fluffy’)) (Is ?p ’pizza’)) (or (not (Is ?d ’dog’)) (not (HasMaterial ?d ’fluffy’)) (RightOf ?p ?d)))))



VLM Output:

```
{
    "satisfies_constraint": false,
    "satisfied_clauses": [
        "(IsStyle 'cartoon')",
        "(or (not (Is ?d 'dog')) (not (HasMaterial ?d 'fluffy')) (Is ?p 'pizza'))"
    ]
}
```

Reason: There is a fluffy dog that, however, has no pizza to its right.

E. Prompts

E.1. Prompts for Instruction Generation

Generation Prompt in Natural Scenario

```
You are an expert proficient in First-Order Logic (FOL) and Computer Vision scene
↪  synthesis. Your core task is to generate high-quality data pairs describing
↪  visual scenes based on a given vocabulary and logical constraints. Your input
↪  and output must be in strict JSON format.

#### Core Concept Definitions

*   **Dependency Closure**: A DSL expression can be decomposed into multiple
↪  top-level logical units connected by `(and ...)`. A logical closure is a minimal
↪  set of quantified variables connected to each other via binary predicates (such
↪  as `Holding`, `On`, `LeftOf`). Connectivity is transitive.
*   **Dependency Depth**: **Dependency Depth** is defined as **the total number of
↪  quantified variables (introduced by `exists`, `forall`) within a single logical
↪  closure**. The dependency depth of the entire DSL expression is the maximum
↪  dependency depth among all its logical closures.
*   **Scene Style**: Describes the overall artistic style or rendering method of the
↪  image. If a style is specified in the input parameters, the `IsStyle` predicate
↪  must be used in the DSL to declare it.

#### DSL Domain Definition
```

```
You must strictly adhere to the complete syntax and vocabulary of the following DSL
↪  and cannot create words outside the vocabulary. In our setting, all directions
↪  are relative to the frame.
{dsl}


#### Few-shot Examples

Below are compliant examples. Please pay attention to their JSON structure,
↪  especially the use of predicates and style control.

**[Example 1: Basic Attributes and Interaction (Holding, Touching, HasState)]**
*   **Generation Parameters:**
{{
  "contained_vocabulary": ["woman", "apple", "basket"],
  "style_control": {{
    "target_style": "oil painting"
  }},
  "target_dependency_depth": {{
    "num_object_class": 3,
    "regular": 3,
    "count": 0
  }},
  "structure_control": {{
    "num_closures": 1,
    "closure_depths": [3],
    "exists_forall_struct": [["exists", "exists", "exists"]]
  }},
  "predicate_control": {{
    "required_predicates": ["Holding", "HasState", "HasColor", "IsStyle"]
  }}
}}

*   **Output:**
{{
  "natural_language_description": "This is an oil painting style image. The scene
  ↪  features a woman holding a red apple. The apple is whole. The woman is also
  ↪  holding a basket.",
  "dsl_representation": "(and (IsStyle 'oil painting') (exists ?w (exists ?a (exists
  ↪  ?b (and (Is ?w 'woman') (Is ?a 'apple') (Is ?b 'basket') (HasColor ?a 'red')
  ↪  (HasState ?a 'whole') (Holding ?w ?a) (Holding ?w ?b))))))"
}}

**[Example 2: Spatial Containment and Material (In, On, HasMaterial, HasShape)]**
*   **Generation Parameters:**
{{
  "contained_vocabulary": ["table", "bowl", "spoon", "glass"],
  "style_control": {{
    "target_style": null
  }},
  "target_dependency_depth": {{
    "num_object_class": 4,
    "regular": 4,
    "count": 0
  }},
  "structure_control": {{
    "num_closures": 1,
    "closure_depths": [4],
    "exists_forall_struct": [["exists", "exists", "exists", "exists"]]
  }},
  "predicate_control": {{
    "required_predicates": ["On", "HasMaterial", "HasShape"]
  }}
}}
```

```
*   **Output:**
{{
  "natural_language_description": "A round bowl sits on a wooden table. Inside the
  ↪  bowl is a metal spoon. There is also a glass cup on the table.",
  "dsl_representation": "(exists ?t (exists ?b (exists ?s (exists ?g (and (Is ?t
  ↪  'table') (HasMaterial ?t 'wooden') (Is ?b 'bowl') (HasShape ?b 'round') (Is ?s
  ↪  'spoon') (HasMaterial ?s 'metal') (Is ?g 'glass') (On ?b ?t) (In ?s ?b) (On ?g
  ↪  ?t))))))"
}}

**[Example 3: Relative Position and Universal Quantifier (LeftOf, RightOf, forall)**
*   **Generation Parameters:**
{{
  "contained_vocabulary": ["car", "bus", "tree"],
  "style_control": {{
    "target_style": "cyberpunk"
  }},
  "target_dependency_depth": {{
    "num_object_class": 3,
    "regular": 3,
    "count": 0
  }},
  "structure_control": {{
    "num_closures": 1,
    "closure_depths": [3],
    "exists_forall_struct": [["exists", "forall", "exists"]]
  }},
  "predicate_control": {{
    "required_predicates": ["LeftOf", "RightOf", "IsStyle"]
  }}
}}

*   **Output:**
{{
  "natural_language_description": "In this image, all directions are relative to the
  ↪  frame. Generate a cyberpunk style image. There is a tree on the left side of
  ↪  the frame. For all cars in the image, they are located to the right of this
  ↪  tree. Additionally, there is a bus located to the left of the tree.",
  "dsl_representation": "(and (IsStyle 'cyberpunk') (exists ?t (and (Is ?t 'tree')
  ↪  (forall ?c (implies (Is ?c 'car') (RightOf ?c ?t))) (exists ?b (and (Is ?b
  ↪  'bus') (LeftOf ?b ?t))))))"
}}

**[Example 4: Vertical Relations and Adjacency (Above, Below, NextTo)]**
*   **Generation Parameters:**
{{
  "contained_vocabulary": ["lamp", "desk", "chair", "rug"],
  "style_control": {{
    "target_style": "cartoon"
  }},
  "target_dependency_depth": {{
    "num_object_class": 4,
    "regular": 4,
    "count": 0
  }},
  "structure_control": {{
    "num_closures": 1,
    "closure_depths": [4],
    "exists_forall_struct": [["exists", "exists", "exists", "exists"]]
  }},
  "predicate_control": {{
    "required_predicates": ["Above", "NextTo", "IsStyle"]
  }}
```

```
}}

*   **Output:**
{{
  "natural_language_description": "In this image, all directions are relative to the
  ↪  frame. This is a cartoon style scene. A lamp hangs above a desk. Next to the
  ↪  desk is a chair. A rug is located below both the chair and the desk.",
  "dsl_representation": "(and (IsStyle 'cartoon') (exists ?l (exists ?d (exists ?c
  ↪  (exists ?r (and (Is ?l 'lamp') (Is ?d 'desk') (Is ?c 'chair') (Is ?r 'rug')
  ↪  (Above ?l ?d) (NextTo ?c ?d) (Below ?r ?c) (Below ?r ?d)))))))"
}}

**[Example 5: Multiple Closures and Negative Logic (not, HasState)]**
*   **Generation Parameters:**
{{
  "contained_vocabulary": ["door", "cat", "sofa"],
  "style_control": {{
    "target_style": null
  }},
  "target_dependency_depth": {{
    "num_object_class": 3,
    "regular": 2,
    "count": 0
  }},
  "structure_control": {{
    "num_closures": 2,
    "closure_depths": [1, 2],
    "exists_forall_struct": [["exists"], ["exists", "exists"]]
  }},
  "predicate_control": {{
    "required_predicates": ["HasState", "not", "On"]
  }}
}}

*   **Output:**
{{
  "natural_language_description": "There is a door that is closed. Furthermore,
  ↪  there is a cat that is not on the sofa.",
  "dsl_representation": "(and (exists ?d (and (Is ?d 'door') (HasState ?d
  ↪  'closed'))) (exists ?c (exists ?s (and (Is ?c 'cat') (Is ?s 'sofa') (not (On
  ↪  ?c ?s))))))"
}}

---
#### Your Task

Now, please generate a **brand new**, **logical** scene description data pair
↪  strictly following the JSON format and the parameters specified below.
Please pay special attention to:
1.  **Style Handling**: If `style_control.target_style` is `null`, do not generate
↪  the `(IsStyle ...)` predicate, and do not mention style in the natural language
↪  description. Otherwise, it must be included.
2.  **Predicate Constraints**: You can only use predicates existing in the DSL
↪  definition, and must prioritize using predicates specified in
↪  `required_predicates`. If using only `required_predicates` cannot generate a
↪  natural sample, you may appropriately add suitable predicates.
3.  **Vocabulary Constraints**: You can only use object types existing in the DSL
↪  definition, and must prioritize using vocabulary specified in
↪  `contained_vocabulary`. If a predicate like `Holding` exists, but
↪  `contained_vocabulary` does not contain an entity capable of this action (e.g.,
↪  only containing inanimate objects), you may add an entity yourself.
```

```
4.  **Spatial Relation Annotation**: If your generated scene involves any spatial
↪  relationship predicates (e.g., LeftOf, RightOf, Above, Below, On, In, NextTo),
↪  you must insert the sentence "In this image, all directions are relative to the
↪  frame." to the start of the natural_language_description to provide clear
↪  context.

**Important**
In summary, the highest guideline for data generation is to ensure the generated
↪  data is reasonable, natural, and free of unnatural descriptions.

*   **Generation Parameters:**
{instruction}

*   **Output:**
{{
  "natural_language_description": "...",
  "dsl_representation": "..."
}}

Please directly output a complete, comment-free JSON object, and do not include any
↪  additional explanation or Markdown code block markers.
```

Generation Prompt in Knolling Scenario

```
You are an expert proficient in First-Order Logic and Computer Vision data
↪  annotation. Your core task is to generate high-quality data pairs for evaluating
↪  advanced text-to-image models on their "structured compositional generalization
↪  capabilities" under the **Knolling (neat arrangement of items from an overhead
↪  view) style**. Both your input and output must be in strict JSON format.

#### Core Concept Definitions

*   **Dependency Closure**: A DSL expression can be decomposed into multiple
↪  independent **logical closures** connected by top-level `(and ...)`. A **logical
↪  closure** is defined as a minimal set of quantified variables formed by
↪  interlinking through binary predicates (such as `RightOf`, `Above`, `NextTo`).
↪  This association is **transitive**: if variable `?a` and `?b` are associated, and
↪  `?b` and `?c` are associated, then `?a`, `?b`, and `?c` belong to the same
↪  closure.
*   **Dependency Depth**: **Dependency Depth** is defined as **the total number of
↪  quantified variables (introduced by `exists`, `forall`) within a single logical
↪  closure**. The dependency depth of the entire DSL expression is the maximum
↪  dependency depth among all its logical closures.

#### DSL Domain
You must strictly adhere to the complete syntax and vocabulary of the following DSL.
{dsl}

#### Few-shot Examples

Below are compliant examples. Please study their JSON structure and content.

**[Knolling Style Example 1: Basic Spatial Chain]**
*   **Generation Parameters:**
{{
  "contained_vocabulary": ["laptop", "keyboard", "mouse"],
  "target_dependency_depth": {{
    "num_object_class": 3,
    "regular": 3,
    "count": 0
  }},
  "structure_control": {{
```

```
    "num_closures": 1,
    "closure_depths": [3],
    "exists_forall_struct": [["exists", "exists", "exists"]]
  }},
  "predicate_control": {{
    "required_predicates": ["RightOf", "AlignedHorizontally", "Has", "Is"]
  }}
}}

*   **Output:**
{{
  "natural_language_description": "Generate a Knolling-style image. The image
  ↪  contains a white laptop, a black keyboard, and a black mouse. The keyboard is
  ↪  located to the right of the laptop, the mouse is located to the right of the
  ↪  keyboard, and these three objects are aligned horizontally.",
  "dsl_representation": "(exists ?laptop (exists ?keyboard (exists ?mouse (and (Is
  ↪  ?laptop 'laptop') (Has ?laptop 'white') (Is ?keyboard 'keyboard') (Has
  ↪  ?keyboard 'black') (Is ?mouse 'mouse') (Has ?mouse 'black') (RightOf ?keyboard
  ↪  ?laptop) (RightOf ?mouse ?keyboard) (AlignedHorizontally ?laptop ?keyboard)
  ↪  (AlignedHorizontally ?keyboard ?mouse)))))"
}}

**[Knolling Style Example 2: Universal Quantifier Constraint]**
*   **Generation Parameters:**
{{
  "contained_vocabulary": ["wrench", "screwdriver"],
  "target_dependency_depth": {{
    "num_object_class": 2,
    "regular": 2,
    "count": 0
  }},
  "structure_control": {{
    "num_closures": 1,
    "closure_depths": [2],
    "exists_forall_struct": [["forall", "exists"]]
  }},
  "predicate_control": {{
    "required_predicates": ["LeftOf", "Has", "Is"]
  }}
}}

*   **Output:**
{{
  "natural_language_description": "Generate a Knolling-style image. For every wrench
  ↪  in the image, there is a red screwdriver to its left.",
  "dsl_representation": "(and (forall ?w (implies (Is ?w 'wrench') (exists ?d (and
  ↪  (Is ?d 'screwdriver') (Has ?d 'red') (LeftOf ?d ?w))))))"
}}

**[Knolling Style Example 3: Vertical Alignment & Comparison]**
*   **Generation Parameters:**
{{
  "contained_vocabulary": ["book", "ruler", "pen", "calculator"],
  "target_dependency_depth": {{
    "num_object_class": 4,
    "regular": 3,
    "count": 0
  }},
  "structure_control": {{
    "num_closures": 2,
    "closure_depths": [3, 1],
    "exists_forall_struct": [["exists", "exists", "exists"], ["exists"]]
  }},
  "predicate_control": {{
```

```
    "required_predicates": ["Above", "AlignedVertically", "LargerThan",
    ↪  "OnRightSide", "Is"]
  }}
}}

*   **Output:**
{{
  "natural_language_description": "Generate a Knolling-style image. The scene
  ↪  contains a book, a ruler, a calculator, and a pen. The book is above the
  ↪  ruler, the ruler is above the pen, and they are aligned vertically. The book
  ↪  is visually larger than the pen. The calculator is located on the right side
  ↪  of the scene.",
  "dsl_representation": "(and (exists ?b (exists ?ruler (exists ?pen (and (Is ?b
  ↪  'book') (Is ?ruler 'ruler') (Is ?pen 'pen') (Above ?b ?ruler) (Above ?ruler
  ↪  ?pen) (AlignedVertically ?b ?ruler) (AlignedVertically ?ruler ?pen)
  ↪  (LargerThan ?b ?pen))))) (exists ?c (and (Is ?c 'calculator') (OnRightSide
  ↪  ?c))))"
}}

**[Knolling Style Example 4: Complex Spatial Chain]**
*   **Generation Parameters:**
{{
  "contained_vocabulary": ["notebook", "coffee_cup", "pen", "keyboard", "mouse"],
  "target_dependency_depth": {{
    "num_object_class": 5,
    "regular": 5,
    "count": 0
  }},
  "structure_control": {{
    "num_closures": 1,
    "closure_depths": [5],
    "exists_forall_struct": [["exists", "exists", "exists", "exists", "exists"]]
  }},
  "predicate_control": {{
    "required_predicates": ["Above", "Below", "RightOf", "AlignedVertically", "Is"]
  }}
}}

*   **Output:**
{{
  "natural_language_description": "Generate a Knolling-style image. The scene
  ↪  features a notebook as the center. Directly above the notebook is a cup of
  ↪  coffee, and directly below is a keyboard; these three are aligned vertically.
  ↪  To the right of the coffee cup is a pen. To the right of the keyboard is a
  ↪  mouse. Additionally, the pen and the mouse also need to be vertically
  ↪  aligned.",
  "dsl_representation": "(exists ?notebook (exists ?coffee_cup (exists ?pen (exists
  ↪  ?keyboard (exists ?mouse (and (Is ?notebook 'notebook') (Is ?coffee_cup
  ↪  'coffee_cup') (Is ?pen 'pen') (Is ?keyboard 'keyboard') (Is ?mouse 'mouse')
  ↪  (Above ?coffee_cup ?notebook) (Below ?keyboard ?notebook) (RightOf ?pen
  ↪  ?coffee_cup) (RightOf ?mouse ?keyboard) (AlignedVertically ?notebook
  ↪  ?coffee_cup) (AlignedVertically ?notebook ?keyboard) (AlignedVertically ?pen
  ↪  ?mouse)))))))"
}}

---
#### Your Task

Now, please strictly follow the JSON format to generate a **brand new**, **logical**
↪  data pair regarding a **Knolling layout** based on the specified parameters
↪  below. Please exercise your creativity and logical reasoning skills; do not
↪  simply repeat or stitch together the above examples.
You may only use the predicates specified in `required_predicates`.
```

```
*   **Generation Parameters:**
{instruction}

*   **Output:**
{{
  "natural_language_description": "...",
  "dsl_representation": "..."
}}

Please output a complete, comment-free JSON object directly. Do not include any
↪  additional explanations or Markdown code block markers.
```

E.2. Prompts for Semantic Check

Check Prompt in Natural Scenario

```
You are a senior analyst proficient in First-Order Logic and visual arts design.
↪  Your task is to review and optimize data pairs of "Natural Language Description
↪  (NL)" and "Domain Specific Language (DSL)".

#### DSL Domain Definition
{dsl}

#### Core Review Principles (STRICT ENFORCEMENT)

**1. Critical Logic & Physics Failures (`is_contradictory` = true):**
Set `is_contradictory: true` (Resample) ONLY if the following occur:

*   **A. Attribute-Object Mismatch (The "Rectangular Lion" Rule):**
    *   **Geometric Hallucinations:** Living beings (lions, people) and complex
    ↪  mechanisms (bicycles, cars) CANNOT be described with simple geometric
    ↪  primitives unless the style is explicitly "Abstract" or "Cubist".
        *   *REJECT:* "A rectangular lion", "A triangular bicycle", "A square
        ↪  horse", "An oval car".
        *   *ACCEPT:* "A rectangular table", "A round ball", "A triangular
        ↪  sandwich".
    *   **Material Absurdity:**
        *   *REJECT:* "A water chair", "A cloud made of iron" (unless surrealism is
        ↪  specified).

*   **B. Scale & Mass Absurdity (The "Holding" Rule):**
    *   **Allowed (Quantity):** A person/animal CAN hold multiple portable items
    ↪  (e.g., "holding every basket", "holding all the apples", "holding a chair").
    ↪  Do not reject based on quantity alone.
    *   **Forbidden (Mass/Immovability):** A person/animal CANNOT hold **Immovable
    ↪  or Massive** objects.
        *   *REJECT:* "A person holding a refrigerator", "A child holding a house",
        ↪  "A man holding a bus", "A woman holding a bed", "A person holding a
        ↪  tree".

*   **C. Impossible Containment (The "Inside" Rule):**
    *   **Solid Objects:** Items cannot be `Inside` solid organic/inorganic objects.
        *   *REJECT:* "A key inside an apple", "A laptop inside a horse", "A book
        ↪  inside a solid rock".
    *   **Size Constraints:** A large object cannot be inside a significantly
    ↪  smaller one.
        *   *REJECT:* "A bicycle inside a basket", "A sofa inside a backpack".

*   **D. Logical Paradoxes:**
    *   Cyclic dependency `(In ?a ?b) and (In ?b ?a)`, or `(LeftOf ?a ?b) and
    ↪  (RightOf ?a ?b)`.
```

```
**2. Alignment and Naturalization Polishing (`is_aligned` &
↪  `corrected_nl_description`):**
Even if the logic is fine, if the NL expression is unnatural or logically
↪  misaligned, set `is_aligned: false` and provide a polished version.

*   **Preserve Logical Scope (Quantifier Order):** Do not alter the logical meaning
↪  by changing the scope of quantifiers (`forall`, `exists`). This is a critical
↪  translation error, not a stylistic choice.
    *   *DSL Logic:* `forall X, exists Y` means "For each X, there is a Y". This
    ↪  allows for a *different* Y for each X.
    *   *Incorrect NL Translation:* "A Y is [relation] to every X". This incorrectly
    ↪  implies `exists Y, forall X` (a single Y for all X's).
    *   *Correct NL Translation:* "Every X has a Y [relation] to it." or "For each
    ↪  X, there is a Y...".

*   **Eliminate Redundant Logic:** Strictly forbid repetitive descriptions like "A
↪  is above B, which means B is below A". Keep it concise.

*   **Remove "Machine Flavor":**
    *   **Strictly Forbidden:** Describing living beings/objects as "whole" (e.g.,
    ↪  "A whole child", "A whole cat"). This is a translation error.
    *   **Phrasing:** Do not translate logic quantifiers literally (e.g., "For all
    ↪  x..."), but ensure their logical scope is preserved as per the rule above.
    ↪  Use natural phrasing like "All trees..." or "Every tree...".

*   **Attribute Integration:** Colors, materials, and states in the DSL (e.g., `on`,
↪  `broken`, `smiling`) must be naturally integrated into the noun phrase rather
↪  than listed separately.
    *   *Bad:* "There is a car. The car is red." -> *Good:* "A red car."

*   **Translation Style:** Must conform to human expression habits, similar to
↪  high-quality prompts for Midjourney or DALL-E (Visual, Descriptive, Fluent).

*   **Important:** If the original description contains the sentence "In this image,
↪  all directions are relative to the frame.", you must preserve it verbatim.

#### Output Format
You must return a strict JSON object:
{{
  "is_contradictory": boolean,
  "is_aligned": boolean,
  "corrected_nl_description": string | null
}}
*Note: If `is_contradictory` is true, set the latter two fields to false and null
↪  respectively.*

---
#### Review Examples

**[Case 1: Geometric Hallucination (Refuse)]**
*   **Input:** "A cyberpunk style scene where every lion is rectangular."
*   **Analysis:** Lions cannot be rectangular. Attribute error.
*   **Output:** {{ "is_contradictory": true, "is_aligned": false,
↪  "corrected_nl_description": null }}

**[Case 2: Mass/Scale Violation (Refuse)]**
*   **Input:** "A child is holding a refrigerator."
*   **Analysis:** A refrigerator is too heavy for a child.
*   **Output:** {{ "is_contradictory": true, "is_aligned": false,
↪  "corrected_nl_description": null }}

**[Case 3: Unnatural/Redundant (Polish)]**
*   **Input:** "There is a red apple on a wooden table. That is to say, the table is
↪  under the red apple."
```

```
*   **Analysis:** Logic is valid, but the second sentence is redundant.
*   - **Output:** {{ "is_contradictory": false, "is_aligned": false,
↪  "corrected_nl_description": "A wooden table with a red apple placed on it." }}

**[Case 4: Machine Flavor/Attribute Integration (Polish)]**
*   **Input:** "For every bird, there is a laptop. The laptop is broken."
*   **Analysis:** "For every" is acceptable but can be more natural. "The laptop is
↪  broken" should be integrated.
*   **Output:** {{ "is_contradictory": false, "is_aligned": false,
↪  "corrected_nl_description": "Next to every bird sits a broken laptop." }}

**[Case 5: Quantifier Scope Misalignment (Polish)]**
*   **Input (DSL Context):** `(forall ?p (plant ?p) (exists ?m (man ?m) (LeftOf ?m
↪  ?p)))`
*   **Input (NL):** "A man is to the left of every plant."
*   **Analysis:** The NL incorrectly implies a *single man* for all plants (`exists
↪  man, forall plant`). The DSL's `forall plant, exists man` scope allows for a
↪  different man for each plant. This is a logical scope misalignment.
*   **Output:** {{ "is_contradictory": false, "is_aligned": false,
↪  "corrected_nl_description": "Every plant has a man to its left." }}

**[Case 6: Valid "Greedy" Quantifier (Accept)]**
*   **Input:** "A woman is holding every basket in the scene."
*   **Analysis:** Baskets are portable. Holding multiple is physically possible. The
↪  NL structure correctly implies `exists woman, forall basket`, which is a valid
↪  logical statement.
*   **Output:** {{ "is_contradictory": false, "is_aligned": true,
↪  "corrected_nl_description": null }}

---
#### Your Task
Please review the following data:
*   **Input:**
{sample_to_check}

*   **Output:**
{{
  "is_contradictory": ...,
  "is_aligned": ...,
  "corrected_nl_description": ...
}}

Please directly output a complete, comment-free JSON object, and do not include any
↪  additional explanation or Markdown code block markers.
```

Check Prompt in Knolling Scenario

```
You are an extremely rigorous first-order logic analyst and data quality
↪  verification expert. Your task is to review a data pair consisting of a natural
↪  language description (NL) and a Domain-Specific Language (DSL) representation to
↪  ensure its logical soundness and the consistency between the two. Your input and
↪  output must both be in strict JSON format.

#### DSL Domain
You must strictly adhere to the complete syntax and vocabulary of the following DSL
↪  for your analysis.
{dsl}

#### Core Review Principles

Your review process consists of two core steps, which must be executed in order:

**1. Contradiction Check:**
```

```
First, you must determine if the DSL expression itself contains an internal
↪  contradiction, making it **unsatisfiable** in any two-dimensional image. Mainly
↪  check for two types of contradictions:
*   **Logical Paradoxes:** For example, using the `forall` quantifier leads to
↪  infinite recursive generation (e.g., 'for every A, there exists a B, and next to
↪  B, there is another A'). This is impossible to realize in a finite image space.
*   **Geometric Paradoxes:** Describing spatial relationships that are physically
↪  impossible to coexist. For example, simultaneously requiring two objects to be
↪  `(RightOf ?a ?b)` and `(AlignedVertically ?a ?b)`.

If any such contradiction is detected, your task ends immediately.

**2. Alignment Check:**
You only proceed to this check if the DSL has **no** internal contradictions. The
↪  core principle of this step is:
**The DSL expression must not introduce any logical or spatial constraints that are
↪  not explicitly stated or strongly implied in the NL.**

In other words, the set of valid scenes defined by the DSL must be a **superset** or
↪  an **equal set** of the scenes described by the NL, and never a **subset**.

*   **Aligned Example:**
    *   NL: "A picture with a red pen and a book."
    *   DSL: `(exists ?p (exists ?b (and (Is ?p 'pen') (Has ?p 'red') (Is ?b
    ↪  'book'))))`
    *   **Analysis:** Aligned.

*   **Misaligned Examples (DSL is more specific than NL):**
    *   NL: "A picture with two books."
    *   DSL: `(exists ?b1 (exists ?b2 (and (Is ?b1 'book') (Is ?b2 'book') (RightOf
    ↪  ?b2 ?b1))))`
    *   **Analysis:** Misaligned. The NL specifies no spatial relationship, but the
    ↪  DSL arbitrarily adds the `(RightOf ?b2 ?b1)` constraint.
    *   NL: "Two laptops and two lighters are arranged in a 2x2 grid."
    *   DSL: `(exists ?l1 .. (and .. (RightOf ?g1 ?l1) (Below ?l2 ?l1) (RightOf ?g2
    ↪  ?l2)))`
    *   **Analysis:** Misaligned. The "2x2 grid" in the NL is ambiguous and allows
    ↪  for multiple specific layouts. However, the DSL specifies only **one** of
    ↪  these layouts, making the DSL's constraints stronger than the NL's.

#### Output Format
You must return a strict JSON object containing the following three fields:
*   `"is_contradictory"` (boolean): `true` if the DSL contains an internal
↪  contradiction, otherwise `false`.
*   `"is_aligned"` (boolean | null):
    *   If `"is_contradictory"` is `true`, this field must be `null`.
    *   If the DSL is not contradictory, determine if the NL and DSL are aligned.
    ↪  `true` for aligned, `false` for misaligned.
*   `"corrected_nl_description"` (string | null):
    *   If `"is_contradictory"` is `true`, or if `"is_aligned"` is `true`, this field
    ↪  must be `null`.
    *   Only when `"is_aligned"` is `false`, this field must contain a **revised,
    ↪  clear, and unambiguous natural language description that perfectly
    ↪  corresponds to the given DSL.**

---
#### Few-shot Review Examples

**[Example 1: Logical Paradox]**
*   **Input:**
{{
```

```
  "natural_language_description": "Generate a Knolling-style image. For every pizza
  ↪  in the image, there exists a game controller to its right, to the right of
  ↪  which is another game controller, and to the right of that game controller is
  ↪  another pizza.",
  "dsl_representation": "(and (forall ?p1 (implies (Is ?p1 'pizza') (exists ?g1
  ↪  (exists ?g2 (exists ?p2 (and (Is ?g1 'game_controller') (Is ?g2
  ↪  'game_controller') (Is ?p2 'pizza') (RightOf ?g1 ?p1) (RightOf ?g2 ?g1)
  ↪  (RightOf ?p2 ?g2))))))))"
}}
*   **Output:**
{{
  "is_contradictory": true,
  "is_aligned": null,
  "corrected_nl_description": null
}}

**[Example 2: Geometric Paradox]**
*   **Input:**
{{
  "natural_language_description": "Generate a Knolling-style image. There is a black
  ↪  calculator, a red tie, and a watch. They are placed in order from left to
  ↪  right, and the calculator and the watch must also be vertically aligned.",
  "dsl_representation": "(exists ?c (exists ?t (exists ?w (and (Is ?c 'calculator')
  ↪  (Has ?c 'black') (Is ?t 'tie') (Is ?w 'watch') (RightOf ?t ?c) (RightOf ?w ?t)
  ↪  (AlignedVertically ?c ?w))))))"
}}
*   **Output:**
{{
  "is_contradictory": true,
  "is_aligned": null,
  "corrected_nl_description": null
}}

**[Example 3: Misaligned]**
*   **Input:**
{{
    "natural_language_description": "Generate a Knolling-style image. At the bottom
    ↪  of the image, there is a horizontally aligned row of items, from left to
    ↪  right: a pen, a watch, a pen, a watch, and a pen. In the left half of the
    ↪  image, for any pan, there is a knife placed in each of the four cardinal
    ↪  directions (above, below, left, right), forming a cross shape. Additionally,
    ↪  there are two laptops and two lighters arranged in a 2x2 grid.",
    "dsl_representation": "(and (exists ?p1 (exists ?w1 (exists ?p2 (exists ?w2
    ↪  (exists ?p3 (and (Is ?p1 'pen') (Is ?w1 'watch') (Is ?p2 'pen') (Is ?w2
    ↪  'watch') (Is ?p3 'pen') (OnBottomSide ?p1) (RightOf ?w1 ?p1) (RightOf ?p2
    ↪  ?w1) (RightOf ?w2 ?p2) (RightOf ?p3 ?w2) (AlignedHorizontally ?p1 ?w1)
    ↪  (AlignedHorizontally ?w1 ?p2) (AlignedHorizontally ?p2 ?w2)
    ↪  (AlignedHorizontally ?w2 ?p3)))))))) (forall ?pan (implies (and (Is ?pan
    ↪  'pan') (OnLeftSide ?pan)) (exists ?k1 (exists ?k2 (exists ?k3 (exists ?k4
    ↪  (and (Is ?k1 'knife') (Is ?k2 'knife') (Is ?k3 'knife') (Is ?k4 'knife')
    ↪  (RightOf ?k1 ?pan) (Below ?k2 ?pan) (LeftOf ?k3 ?pan) (Above ?k4
    ↪  ?pan))))))))) (exists ?l1 (exists ?g1 (exists ?l2 (exists ?g2 (and (Is ?l1
    ↪  'laptop') (Is ?g1 'lighter') (Is ?l2 'laptop') (Is ?g2 'lighter') (RightOf
    ↪  ?g1 ?l1) (Below ?l2 ?l1) (RightOf ?g2 ?l2)))))))"
}}
*   **Output:**
{{
  "is_contradictory": false,
  "is_aligned": false,
```

```
  "corrected_nl_description": "Generate a Knolling-style image. At the bottom of the
  ↪  image, there is a horizontally aligned row of items, from left to right: a
  ↪  pen, a watch, a pen, a watch, and a pen. In the left half of the image, for
  ↪  any pan, there is a knife to its right, a knife below it, a knife to its left,
  ↪  and a knife above it. Additionally, there exists a laptop with a lighter to
  ↪  its right, and below that laptop is another laptop, which also has a lighter
  ↪  to its right."
}}

**[Example 4: Qualified Sample]**
*   **Input:**
{{
  "natural_language_description": "Generate a Knolling-style image. There is a white
  ↪  laptop, and to its right is a black keyboard.",
  "dsl_representation": "(exists ?laptop (exists ?keyboard (and (Is ?laptop
  ↪  'laptop') (Has ?laptop 'white') (Is ?keyboard 'keyboard') (Has ?keyboard
  ↪  'black') (RightOf ?keyboard ?laptop))))"
}}
*   **Output:**
{{
  "is_contradictory": false,
  "is_aligned": true,
  "corrected_nl_description": null
}}

---
#### Your Task

Now, review the following data pair and return your analysis strictly in the JSON
↪  format described above.

*   **Sample to Check:**
{sample_to_check}

*   **Output:**
{{
  "is_contradictory": ...,
  "is_aligned": ...,
  "corrected_nl_description": ...
}}
Directly output a single, complete JSON object without any comments, explanations,
↪  or Markdown code block markers.
```

E.3. Prompts for Evaluation

Evaluation Prompt in Natural Scenario

```
You are a top-tier multimodal AI analyst proficient in First-Order Logic and image
↪  content verification.

**Task:**
Your task is to precisely analyze a given image and perform a dual verification
↪  based on a "Natural Language Constraint" and a structured "Logical Expression".
↪  You need to complete two core tasks:
1.  **Overall Constraint Evaluation (Hard Constraint):** Determine if the image
↪  content **completely** satisfies the overall requirements of the "Natural
↪  Language Constraint".
2.  **Atomic Clause Verification (Soft Constraint):** Analyze every **atomic
↪  clause** in the "Logical Expression" one by one, and filter out the clauses that
↪  hold true in the image.
```

```
---
#### **Input Format Description**
The "Logical Expression" you receive has a specific structure:
1.  **Independent Logical Blocks:** The entire expression consists of one or more
↪  "blocks" that are logically independent of each other. These blocks are connected
↪  at the top level by `and`.
2.  **Quasi-CNF (Quasi-Conjunctive Normal Form):** Inside each regular "logical
↪  block" is a quasi-CNF form: quantifiers (`forall`, `exists`) are placed at the
↪  beginning, followed by a series of **atomic clauses** connected by `and`.
3.  **Atomic Clauses:** These are the smallest units for your "soft constraint"
↪  verification. It is typically a simple predicate call (e.g., `(Is ?x 'cat')`) or
↪  a disjunction `(or ...)`.

---
#### **Analysis Instructions**
0.  **The "Zero-Evidence" Protocol (CRITICAL PRIORITY):**
    *   **Image Integrity Check:** Before analyzing logic, scan the image globally.
    *   **Immediate Fail Condition:** If the image is **solid black, solid white,
    ↪  random noise, severe blur, or completely dark** with no discernible objects,
    ↪  you MUST STOP processing immediately.
    *   **Outcome:** In this case, `satisfies_constraint` MUST be `false`, and
    ↪  `satisfied_clauses` MUST be strictly `[]` (empty list). Do not attempt to
    ↪  hallucinate objects.

1.  **Overall Constraint Evaluation (`satisfies_constraint`):**
    *   First, comprehensively understand the full requirements of the "Natural
    ↪  Language Constraint" and the "Logical Expression".
    *   Carefully observe the image to determine if it **flawlessly satisfies all
    ↪  requirements**. Specifically, colors, materials, states, styles, and spatial
    ↪  relations must be exactly consistent.
    *   If the semantics of the natural language are ambiguous, **you must use the
    ↪  strict definition of the "Logical Expression" as the final judgment
    ↪  standard**.
    *   If fully satisfied, it is `true`; if there is any discrepancy (e.g.,
    ↪  incorrect material, reversed position), it is `false`.

2.  **Atomic Clause Verification (`satisfied_clauses`):**
    *   Traverse **all** **atomic clauses** in the "Logical Expression" (including
    ↪  those inside all independent logical blocks).
    *   Independently judge whether each atomic clause holds true in the image.
    *   Collect the complete strings of all **atomic clauses that hold true** in the
    ↪  image into a list.
    *   **Prohibition of "Vacuous Truth" for `forall`:**
        *   In standard logic, "All unicorns are pink" is True if there are no
        ↪  unicorns (Vacuous Truth).
        *   **IN THIS TASK, YOU MUST REJECT VACUOUS TRUTH.**
        *   A clause starting with `forall ?x` or involving a group check is
        ↪  considered "Satisfied" **ONLY IF** actual instances of `?x` exist in the
        ↪  image AND they meet the condition.
        *   **Rule:** If the subject of a `forall` clause (e.g., "all fruits") is
        ↪  absent from the image, that clause is **FALSE**. Do not include it in
        ↪  the output.


**Special Case:** If the image content is completely unrelated to the scene
↪  described in the "Natural Language Constraint", `satisfies_constraint` should be
↪  `false`, and `satisfied_clauses` should be an empty list `[]`.

---
#### **Example 1**

Suppose the input is as follows:

*   **Image:** An image showing a red apple on a wooden table.
```

```
*   **Natural Language Constraint:** "A red apple is on a metal table."
*   **Logical Expression:**
(exists ?t
  (exists ?a
    (and
      (Is ?t 'table')
      (Is ?a 'apple')
      (On ?a ?t)
      (HasColor ?a 'red')
      (HasMaterial ?t 'metal')
    )
  )
)

**Your analysis process should be:**
1.  **Overall Evaluation:** The natural language requires the table to be "metal",
↪  but the table in the image is "wooden". Therefore, the overall constraint is
↪  **not satisfied**. `satisfies_constraint` is `false`.
2.  **Clause Verification:**
    *   Atomic Clause 1: `(Is ?t 'table')` -> Holds, there is indeed a table in the
    ↪  image.
    *   Atomic Clause 2: `(Is ?a 'apple')` -> Holds, there is indeed an apple in the
    ↪  image.
    *   Atomic Clause 3: `(On ?a ?t)` -> Holds, the apple is indeed on the table.
    *   Atomic Clause 4: `(HasColor ?a 'red')` -> Holds, the apple is red.
    *   Atomic Clause 5: `(HasMaterial ?t 'metal')` -> **Does not hold**, the table
    ↪  in the image looks like wood, not metal.
3.  **Final Output:** Collect all valid atomic clauses.

**The corresponding JSON output should be:**
```json
{{
  "satisfies_constraint": false,
  "satisfied_clauses": [
    "(Is ?t 'table')",
    "(Is ?a 'apple')",
    "(On ?a ?t)",
    "(HasColor ?a 'red')"
  ]
}}
```

Example 2 (The "Black Screen" & Vacuous Truth Case)

*   **Image:** A completely black image (or an image with only static noise).
*   **Natural Language Constraint:** "All elephants in the room are pink and
↪ standing next to an apple."
*   **Logical Expression:**
(forall ?e
  (implies
    (Is ?e 'elephant')
    (and
      (HasColor ?e 'pink')
      (exists ?a (and (Is ?s 'apple') (NextTo ?e ?a)))
)
)
)

Your analysis process should be:
1. **Step 0 Check:** The image is black. There is no visual content.
2. **Constraint Check:** Since nothing is visible, the complex description of
↪ elephants and spaceships is definitely not satisfied. `satisfies_constraint` is
↪ `false`.
```

```
3.  **Clause Verification (Anti-Hallucination):**
    *   Logic says `forall` is True if no elephants exist (Vacuous Truth).
    *   **HOWEVER**, applying the **"Prohibition of Vacuous Truth"**: Since no
    ↪  elephant `?e` is visible in the pixels, the logic verification fails.
    *   No objects can be confirmed.
4.  **Final Output:** Return failure and an empty list.

**The corresponding JSON output should be:**
```json
{{
  "satisfies_constraint": false,
  "satisfied_clauses": []
}}
```

Input:

Natural Language Constraint:
{nl}

Logical Expression:
{cnf}

Output Requirement:
Please return your analysis result strictly in the following JSON format, **without
↪ adding any extra explanation, code markers, or explanatory text.**

```json
{{
  "satisfies_constraint": <true_or_false>,
  "satisfied_clauses": [
    "<Full string of the first satisfied atomic clause>",
    "<Full string of the second satisfied atomic clause>",
    "..."
  ]
}}
```

Grounding Prompt in Knolling Scenario

```
You are an expert visual grounding assistant for a formal verification system.
Your task is to detect objects in the provided image and extract their visual
↪  attributes based strictly on the provided vocabulary.

### VOCABULARY CONSTRAINTS
You must ONLY use terms from the following lists. If an object's attribute is
↪  ambiguous, mixed, or not in the list, you must output "other".

- **Allowed Object Classes**: {object_classes}
- **Allowed Colors**: {colors}

### INSTRUCTIONS
1. **Detection**: Locate all objects visible in the image that belong to the
↪  "Allowed Object Classes" list.
2. **Attribute Extraction**: For each detected object, identify its dominant color
↪  using the lists above.
3. **Bounding Boxes**: Provide the bounding box as [xmin, ymin, xmax, ymax].
   - **IMPORTANT**: The coordinates must be **normalized integers from 0 to 1000**.
   - (0,0) is top-left, (1000,1000) is bottom-right.

### OUTPUT FORMAT
Output a valid JSON object strictly following this structure:
```
```

```
{{
  "objects": [
    {{
      "label": "One of {object_classes2}",
      "color": "One of {colors2}",
      "box_2d": [xmin, ymin, xmax, ymax]
    }},
    ...
  ]
}}
```

E.4. Prompts for Style Control

Knolling Style (Full)

```
[Generation Rules and Style Definition]
1. Art Style: Please generate a standard Knolling style (flat lay organization)
↪  image.
   - Perspective: Must be a strictly vertical 90-degree top-down / overhead view.
   - Layout: Objects should be neatly arranged with space between them. Overlapping
   ↪  or occlusion of objects is strictly forbidden.
   - Background: A clean, flat background (white, light gray, or pale yellow is
   ↪  recommended).

2. Spatial Position Definition (Crucial):
   To ensure proper spacing and avoid overlap, positional descriptions are
   ↪  determined by the relative relationship between an **object's boundary and a
   ↪  reference object's center point**:

   - [On the left side / right side / top / bottom]: Refers to the object's
   ↪  geometric center being located in the corresponding half of the entire frame.
   - [In the Center (InCenter)]: Refers to the object's center point being strictly
   ↪  within the central cell of a 3x3 grid dividing the frame.

   **Relative Position Determination Logic:**
   - [A is to the left of B (LeftOf)]: The X-coordinate of A's **right edge** must
   ↪  be less than the X-coordinate of B's **center point**.
   - [A is to the right of B (RightOf)]: The X-coordinate of A's **left edge** must
   ↪  be greater than the X-coordinate of B's **center point**.
   - [A is above B (Above)]: The Y-coordinate of A's **bottom edge** must be less
   ↪  than the Y-coordinate of B's **center point** (visually, A's bottom is higher
   ↪  than B's center).
   - [A is below B (Below)]: The Y-coordinate of A's **top edge** must be greater
   ↪  than the Y-coordinate of B's **center point** (visually, A's top is lower
   ↪  than B's center).

   - [Horizontally Aligned]: The Y-coordinates of the objects' center points are
   ↪  nearly identical.
   - [Vertically Aligned]: The X-coordinates of the objects' center points are
   ↪  nearly identical.

3. Object Attributes:
   - Please ensure object color descriptions are consistent.
   - Size Relations: If 'A is larger than B' is described, ensure the projected area
   ↪  of A in the frame is greater than that of B.

[The specific description is as follows]:
```

Knolling Style (Simplified)

```
Generate a clean knolling-style flat lay image.

- Perspective: strict 90-degree top-down view.
- Layout: all objects are neatly arranged with clear spacing between them.
  No overlapping or touching objects.
- Background: solid, clean background (white, light gray, or pale yellow).

- Composition:
  - Objects are placed in clear rows or columns.
  - Relative positions are obvious (left / right / above / below).
  - Objects that are described as aligned should appear visually aligned.

- Object rules:
  - Colors are consistent with the description.
  - Size relationships are respected (larger objects clearly appear larger).

[Scene description:]
```

Knolling Style (LLM)

```
**[Role and Task Definition]**

You are an AI Scene Planner serving a text-to-image generation system. Your core
↪  task is to accurately convert a user's natural language scene description into a
↪  structured JSON object. This JSON object defines the attributes, precise 2D
↪  bounding boxes, and spatial relationships of every object in the scene. You must
↪  strictly follow all the rules and logic defined below to generate this JSON.

**[1. Canvas and Coordinate System]**

1.  **Canvas Size**: All calculations are based on a virtual `1000x1000` pixel
↪  canvas.
2.  **Coordinate Origin**: The coordinate system's origin `(0, 0)` is at the top-left
↪  corner of the canvas.
3.  **Coordinate Range**: All `box_2d` coordinate values (x_min, y_min, x_max, y_max)
↪  must be integers within the range `[0, 999]`.

**[2. Output Format]**

Your output must be a single, well-formatted JSON object, without any explanatory
↪  text outside of the JSON itself. The structure is as follows:

```json
[
    {{
        "label": "object_label",
        "color": "object_color",
        "box_2d": [x_min, y_min, x_max, y_max]
    }}
]
```

[3. Core Principles of Art Style and Layout: Knolling]

The layout of all objects must adhere to the core principles of the Knolling style:

1. **Perspective**: Must be a strictly vertical 90-degree top-down / overhead view.
2. **Layout**: Objects should be neatly arranged. Most importantly, **the bounding
↪ boxes (`box_2d`) of any two objects must never overlap**. There must be clear
↪ spacing between all objects.
```

```
3.  **Background**: The background is a solid color. This does not affect the JSON
↪  generation for the objects, but you should be aware that objects are placed
↪  independently.

**[4. Spatial Position Definition (Crucial Logic)]**

These are the most critical rules you must follow when generating `box_2d`
↪  coordinates.

**4.1. Basic Definitions**

-   The bounding box of an object `A` is `[A_xmin, A_ymin, A_xmax, A_ymax]`.
-   The coordinates of the center point `A_center` of an object `A` are:
    -   `A_center_x = (A_xmin + A_xmax) / 2`
    -   `A_center_y = (A_ymin + A_ymax) / 2`

**4.2. Absolute Position Definition (based on object center point)**

-   **[On the left side]**: The object's `center_x` < 500.
-   **[On the right side]**: The object's `center_x` > 500.
-   **[On the top side]**: The object's `center_y` < 500.
-   **[On the bottom side]**: The object's `center_y` > 500.
-   **[In the center]**: The object's center point `(center_x, center_y)` must be
↪  strictly within the central area of a 3x3 grid dividing the canvas, i.e., `333 <
↪  center_x < 666` and `333 < center_y < 666`.

**4.3. Relative Position Determination Logic (based on "object boundary" and
↪  "reference object center point")**

This is the **sole standard** for determining the relative positions between
↪  objects:

-   **[A is to the left of B]**: A's **right edge** (`A_xmax`) must be less than B's
↪  **center point X-coordinate** (`B_center_x`).
-   **[A is to the right of B]**: A's **left edge** (`A_xmin`) must be greater than
↪  B's **center point X-coordinate** (`B_center_x`).
-   **[A is above B]**: A's **bottom edge** (`A_ymax`) must be less than B's **center
↪  point Y-coordinate** (`B_center_y`).
-   **[A is below B]**: A's **top edge** (`A_ymin`) must be greater than B's **center
↪  point Y-coordinate** (`B_center_y`).

**4.4. Alignment**

-   **[Horizontally Aligned]**: The `center_y` coordinates of multiple objects should
↪  be nearly identical.
-   **[Vertically Aligned]**: The `center_x` coordinates of multiple objects should
↪  be nearly identical.

**[5. Learning from an Example]**

**User Description**: "Generate a Knolling style image. There is a black phone in
↪  the center of the frame, another phone on the left side, and a blue pen on the
↪  right side."

**Your Correct Output**:
```json
[
    {{
        "label": "phone",
        "color": "white",
        "box_2d": [84, 232, 337, 759]
    }},
    {{
        "label": "phone",
```
```

```
        "color": "black",
        "box_2d": [405, 242, 668, 759]
    }},
    {{
        "label": "pen",
        "color": "blue",
        "box_2d": [827, 249, 874, 762]
    }}
]
```

[6. Object and Color Vocabularies]

The object labels you can use in your output are: {objs}

The color labels you can use in your output are: {colors}

[Begin Task]

Now, strictly following all the rules above, generate the corresponding JSON output for the following user description.

User Description: