SchemaGUI: A Schema-Driven Benchmark for Controllable GUI Generation Evaluation

Anonymous ACL submission

Abstract

Large language models (LLMs) have demonstrated strong potential in graphical user interface (GUI) generation, but reliable evaluation remains challenging due to uncontrolled data distributions, noisy annotations, and limited layout scenario coverage. To address this, we propose SchemaGUI, a template-based benchmark for controllable GUI generation evaluation. By synthesizing paired natural language instructions and deterministic function-call references from parameterized interface schemas, SchemaGUI can generate thousands of zero-noise tasks in seconds entirely independent of human labeling. Based on 1,000 evaluated samples across six representative bilingual scenarios, we benchmark mainstream models including the Qwen3.5 family, Llama-4, and DeepSeek-R1. Our extensive analysis reveals three key insights. First, precise geometric spatial control has replaced instruction following as the primary bottleneck; while scaling Qwen3.5 from 4B to 27B improves structural feasibility from 89.33% to 96.98%, exact geometric alignment improves only marginally (from 75.47% to 80.04%). Second, generation difficulty is highly sensitive to layout complexity, with current LLMs excelling at simple sequential arrangements but suffering severe coordinate drift in dense grids and multi-region compositions.

1 Introduction

Large Language Models (LLMs) have achieved substantial progress in Graphical User Interface (GUI) generation. Early frameworks, such as pix2code (Beltramelli, 2017) and Sketch2Code (Robinson, 2019), demonstrated the feasibility of end-to-end generation by translating screenshots or hand-drawn sketches into code.With the advent of modern multimodal models, approaches like Design2Code (Si et al., 2024) and LayoutGPT (Feng et al., 2023) have pushed the generation targets toward high-fidelity web pages and layouts, while WAFFLE (Liang et al., 2026), ScreenCoder (Jiang et al., 2025), and UICopilot (Gui et al., 2025) have advanced structural alignment, modular reasoning, and functional synthesis. Consequently, generated outputs have evolved from visually similar mockups to fully structured, interactive interfaces that can be rendered in a browser.

Evaluating the GUI generation model requires a benchmark dataset that satisfies two fundamental criteria: comprehensive scenario coverage and clean individual samples.As illustrated in Figure 1,current benchmarks fall short on both counts.One approach relies on web crawling and manual annotation, which produces uncontrolled samples, limited scenario diversity, and high annotation costs, making large-scale evaluation uncontrolled. Furthermore, the high manual annotation costs make scaling this approach unsustainable (Gu et al., 2024). However, these synthetic samples inherit model hallucinations and biases, leading to unreliable evaluation results.Another approach uses LLMs to generate data at scale, but the synthetic samples inherit model noise, hallucinations, and biases, making the evaluation targets inherently unclean.This unstable data quality and coverage ultimately make the model comparisons unreliable.What GUI evaluation lacks is a dataset that is coverage-controlled, clean per sample, and scalable.

To address this, we propose SchemaGUI, a template-based framework for automatically generating GUI evaluation data. SchemaGUI covers six common interface types, including calculator, login form, music player, document editor, image editor, and information card, with adjustable layouts, components, and text. Each configuration produces two paired outputs: a natural language task description and a component-wise annotation listing positions, sizes, and text. Using this method, we generated

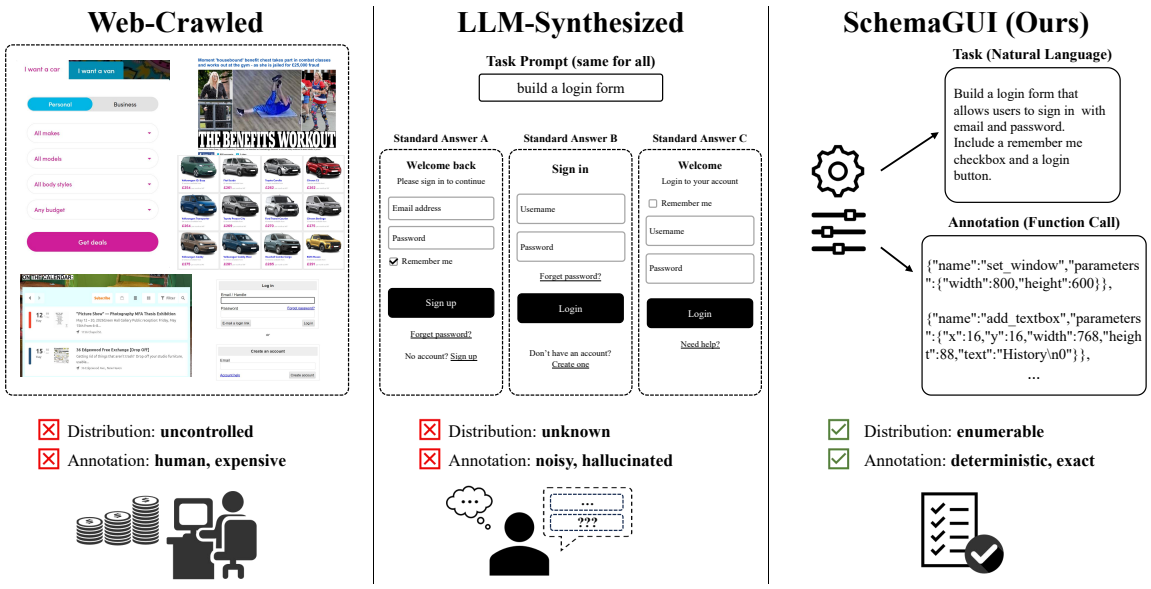

Figure 1: Comparison of different data construction paradigms for GUI generation evaluation. (Left) **Web-Crawled**: data distribution is uncontrolled with expensive manual annotations. (Middle) **LLM-Synthesized**: the data distribution remains unknown while outputs suffer from model noise and hallucinated references. (Right) **SchemaGUI (Ours)**: parametric slot configurations enable enumerable distributions and deterministic, exact annotations.

a dataset of 1,000 tasks, with bilingual (Chinese-English) instructions and cross-backend validation. Models are asked to produce outputs in the same structured format, allowing line-by-line comparison with the reference. The process can generate more than 10,000 samples in seconds, with all annotations derived directly from the template, without relying on human labeling or model outputs.

Based on this dataset, we conducted a systematic evaluation of mainstream large language models, including Qwen 3.5, Llama 4, and DeepSeek R1. The experimental results reveal three key insights. First, **precise geometric spatial control has replaced instruction following as the core bottleneck of current models**. As the model size scales from Qwen3.5-4B to 27B, the structural feasibility of its outputs significantly jumps from 89% to 97%, but the geometric alignment score for component positions and sizes only slowly increases from 75% to 80%, and text compliance persistently stagnates at approximately 84.5%. This indicates that while models are already highly capable of comprehending instructions and generating valid structured formats, they still struggle to achieve precise 2D coordinate planning in the absence of visual feedback. Second, **generation difficulty is highly sensitive to layout complexity**. Existing models excel at simple linear layouts but experience a precipitous drop in performance in scenarios involving dense components (e.g., calculators) or multi-region combinations. This demonstrates that current models are highly susceptible to losing coordinate control amidst repetitive and dense component arrangements. Finally, **long-form reasoning exhibits a negative impact under strict GUI layout constraints**. In dense grid layout tasks, enabling the reasoning mode causes the score of Qwen3.5-27B to plummet by 25.56%. This suggests that although long-form reasoning is highly valuable for long-horizon logical deduction, the redundant intermediate content it generates disrupts the structural stability of function-calls; consequently, for short-horizon spatial layout tasks that require parallel planning and precise positioning, excessive reasoning paradoxically becomes an unnecessary burden.

2 Related Work

2.1 Text-to-UI Generation and End-to-End Models

The development of modern GUI generation techniques originates from early neural layout reconstruction methods. Representative works include pix2code (Beltramelli, 2017), which transforms GUI screenshots into structured code, and Sketch2Code(Robinson, 2019), which further en-

ables the automatic generation of HTML from hand-drawn paper UI sketches.

Building upon these foundations, high-fidelity synthesis has emerged via MLLMs. For instance, Generative UI (Leviathan et al., 2026) highlights that LLMs are effective zero-shot UI generators,and Design2Code (Si et al., 2024) establishes a comprehensive framework for evaluating the translation of visual designs into production-ready code.Beyond direct mapping, recent research focuses on structural alignment and task flexibility. WAFFLE (Liang et al., 2026) introduces a structure-aware attention mechanism to better align UI images with HTML hierarchical logic. Meanwhile, UI-UG (Yang et al., 2025) unifies diverse tasks, including image-to-UI and text-to-code, under a single instruction-following framework. PSD2Code (Chen and Chen, 2025) explores direct functional mapping from structured design files to front-end logic.

However, these end-to-end paradigms rely on stochastic generation without explicit structural constraints, often causing layout drift or component overlapping. To ensure geometric determinism, we introduce a schema-constrained functional intermediate representation (IR).

2.2 Intermediate Representations and Structured Specification

To enhance controllability, researchers have explored structural modeling of UI hierarchies. Early probabilistic frameworks like LayoutVAE (Jyothi et al., 2021) propose VAE-based architectures for stochastic scene layout generation, capturing both object counts and spatial relationships within UI structures. Similarly, HSSN (Li et al., 2022) utilizes multi-label hierarchy constraints and margin-induced embedding refinement to enforce tree-structured dependencies.

To further constrain the generative process, UI grammar (Lu et al., 2023) guides LLMs in maintaining rigorous syntactic consistency during layout synthesis.To rectify latent layout deficiencies, AlignUI (Liu et al., 2026) employs preference-constrained modeling and structural alignment strategies to enhance generative stability. Regarding design abstractions, SpecifyUI (Chen et al., 2025) introduces structured specifications to support the iterative expression of design intent. Recent critique-based frameworks like UICrit (Duan et al., 2024) further emphasize the importance of structural feedback.

However, current models often mimic design patterns without understanding physical layout rules, leading to geometric conflicts like component overlapping. We address this by incorporating a geometric audit on predicted function calls to ensure physical feasibility.

2.3 Programmatic Generation and Functional Synthesis

To move beyond static structural modeling, recent advancements focus on the execution-oriented logic of interfaces. This shift was catalyzed by benchmarks like META-GUI (Sun et al., 2022), which frame UI tasks as multimodal programmatic action sequences across conversational interactions. WebArena (Zhou et al., 2024) establishes a self-hosting web environment to evaluate the ability of agents to execute complex, multi-step programmatic workflows.

To handle the escalating complexity of these tasks, recent frameworks have introduced modular reasoning strategies. GUIDE (Kolthoff et al., 2025) addresses reliability through task decomposition, breaking down high-level UI requirements into manageable sub-goals. CANVAS (Jeong et al., 2025) evaluates models within a tool-based design paradigm, focusing on the iterative use of external software to refine design intent. These advancements have fostered emerging synthesis frameworks, where UICopilot (Gui et al., 2025) leverages context-aware code synthesis to bridge the gap between design and development, and Widget2Code (Zhang et al., 2026) utilizes component-level function calling to map visual widgets directly into executable code.

Nevertheless, these atomized methods often focus on localized tasks rather than maintaining the systemic integrity of the entire layout. We bridge this gap by proposing a unified functional synthesis paradigm that governs the full layout through formal parameter space $\mathcal{S}$ modeling.

3 Methodology

3.1 Formalization of the Schema-driven GUI Design Space

To address the low efficiency of acquiring high-quality data, we formalize the GUI generation task within a controllable and strictly standardized framework.

In our framework, we conceptualize a Graphical User Interface (GUI) as a structured collec-

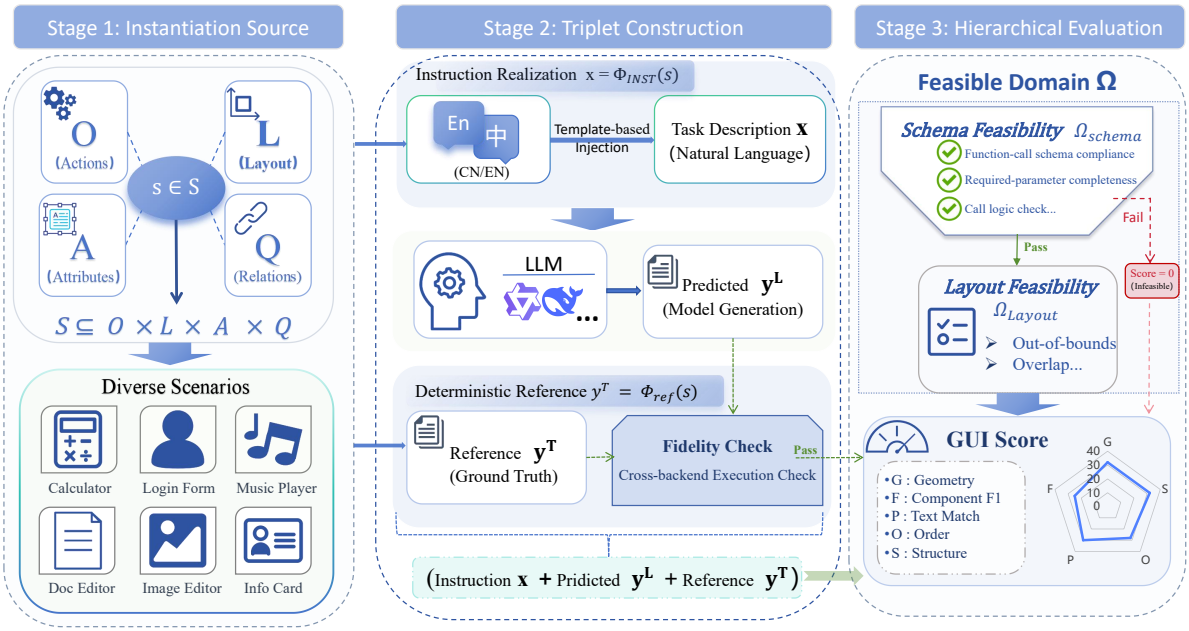

Figure 2: Overview of the SchemaGUI benchmarking. Stage 1 instantiates a GUI instance s from the schema-driven design space $\mathcal{S}$, covering six representative scenarios with controllable actions, layouts, attributes, and relations. Stage 2 constructs an evaluation triplet by realizing s into a natural-language instruction x, collecting the model prediction y^L, and deterministically generating the reference layout y^T. Stage 3 evaluates each prediction through schema and layout feasibility checks, then computes the GUI Score over geometry, component F1, text match, order, and structure.

tion of components, where the generation process is defined as a sequence of atomic function calls. Specifically, we introduce a functional Intermediate Representation (IR) defined as a sequence of atomic function calls y:

$$y = [c_1, c_2, \ldots, c_n]. \tag{1}$$

Each individual call $c_i = (f_i, \theta_i)$ serves as an independent generative instruction.The operation primitive f_i specifies the component type, such as an add button or input,while the associated parameters θ_i provide the specific details. These parameters θ_i consist of geometric attributes $\mathcal{L}$,defining coordinates (x, y) and dimensions (w, h), and Content Attributes $\mathcal{A}$, covering text labels and data values.

Each individual call c_i is independent. However, simply stacking valid calls does not automatically result in a functional GUI.To control the generation of these sequences y, we introduce the Schema-driven Design Space:

$$\mathcal{S} \subseteq \mathcal{O} \times \mathcal{L} \times \mathcal{A} \times \mathcal{Q} \tag{2}$$

where each design instance $s \in \mathcal{S}$ is derived from the global Design Space $\mathcal{S}$ through an exhaustive traversal process. Formally, each instance s is represented as a combination of four functional domains:

Operational Actions ($\mathcal{O}$) represent the actions of the construction process, covering environment initialization and component instantiation. Layout Geometry ($\mathcal{L}$) specifies spatial placement and sizing, defining precise coordinates and structural patterns such as grids. Content Attributes ($\mathcal{A}$) provide the actual content for the components, filling slots with semantic data such as button labels or display text. Qualitative Relations ($\mathcal{Q}$) enforce the rules of organization, governing component grouping and alignment to ensure structural consistency.

Building upon the formalized instance s, the next stage of our framework is to translate it into two distinct forms: a model-readable input and a verifiable reference.

$$x = \Phi_{\text{inst}}(s), \tag{3}$$

$$y^T = \Phi_{\text{ref}}(s), \tag{4}$$

$$y^L = \Psi_{\text{LLM}}(x). \tag{5}$$

Instruction Realization ($s \mapsto x$): The instance s is materialized into a natural language task description x through template-based injection, serving as the input for the model.

Reference Realization ($s \mapsto y^T$): Simultaneously, s is deterministically expanded into a reference sequence y^T following strict procedural rules.

With the instruction x as input, the model generates a predicted sequence y^L. To ensure a fair assessment, we synchronize this prediction with its specific parent instance to form the evaluation triplet (x, y^T, y^L).

3.2 Schema-Conditioned GUI Evaluation

Building upon the design space formalized in Section 3.1, we now describe the evaluation of generated GUI artifacts.

We define the Feasible Domain (Ω) as the admission boundary for evaluating LLM outputs. To prevent "ghost scores" from non-functional artifacts, our assessment follows a two-stage hierarchy:

$$\Omega = \Omega_{\text{schema}} \cap \Omega_{\text{layout}}. \tag{6}$$

Schema feasibility Ω_{schema} measures the model's foundational instruction-following ability through structural executability. Specifically, outputs that violate the schema by providing invalid call-sequence formats or missing required arguments are treated as a failure to adhere to the construction rules and are assigned a score of zero.

Layout feasibility Ω_{layout} evaluates geometric quality for structurally valid outputs using continuous penalties rather than hard rejection. This includes out-of-bounds behavior, where elements exceed visible window boundaries; overlap behavior, where components obstruct each other's visibility; and coordinate-size deviations that reflect precision errors relative to the ground truth y^T.

For output sequences satisfying Ω_{schema}, the GUI Score (GS) provides a holistic assessment by comparing each prediction y_i^L to its reference y_i^T across five complementary dimensions:

Geometry ($\mathcal{G}_i$): geometric fidelity, combining coordinate and size consistency with penalties for out-of-bounds placement and severe overlap behavior. **Component F1** ($\mathcal{F}_i$): set-level completeness and correctness of instantiated components. **Text Match** ($\mathcal{P}_i$): semantic consistency of textual fields and data-driven attributes. **Order** ($\mathcal{O}_i$): call-order consistency measured by the Longest Common Subsequence. **Structure** ($\mathcal{S}_i$): structural quality and adherence to syntax-level validity constraints.

To operationalize the layout penalty, we explicitly quantify the overlap severity among components within the generated interface. For any pair of distinct predicted boxes $b_p, b_q \in y^L$, where p and q denote their respective indices in the output sequence ($p \neq q$), we define the overlap ratio OR as:

$$\text{OR}(b_p, b_q) = \frac{\text{Area}(b_p \cap b_q)}{\min\left(\text{Area}(b_p), \text{Area}(b_q)\right)}. \tag{7}$$

and classify an overlap as severe when $\text{OR}(b_p, b_q) > 0.5$. The resulting overlap and out-of-bound penalties are integrated into the geometry component $\mathcal{G}$, together with coordinate and size consistency terms, yielding a strict yet continuous quality signal.

The final score GS is defined as:

$$\text{GS}_i = \begin{cases} \sum_k w_k m_{i,k}, & y_i^L \in \Omega_{\text{schema}}, \\ 0, & \text{otherwise.} \end{cases} \tag{8}$$

where $\mathbf{m}_i = [\mathcal{G}_i, \mathcal{F}_i, \mathcal{P}_i, \mathcal{O}_i, \mathcal{S}_i]$ is the sub-metric vector."otherwise" denotes a total failure in structural executability due to schema violations.

As summarized in Table 1, we define the default weights $\mathbf{w}$ following a usability-priority principle. We set the weight vector as:

Metric	Notation	Weight
Geometry	$\mathcal{G}$	0.35
Component F1	$\mathcal{F}$	0.25
Text Match	$\mathcal{P}$	0.20
Order	$\mathcal{O}$	0.15
Structure	$\mathcal{S}$	0.05
Total	–	**1.00**

Table 1: Sub-metrics weights in GUI Score GS.

Detailed ablation studies regarding these weight settings are provided in Appendix A.1.

4 Experiment Setup

4.1 Baseline Models

We evaluate multiple LLM baselines across model families and parameter scales. The evaluated set includes Qwen variants (`Qwen3.5-4B`, `Qwen3.5-9B`, `Qwen3.5-27B`) (Qwen Team, 2026), DeepSeek-R1 variants (Guo et al., 2025), and Llama-family instruction models (`Llama-4-Scout-17B-16E-Instruct`) (Adcock et al., 2026).

We use the deterministic template output y^T as a non-LLM reference. It provides the deterministic reference layout against which model predictions y^L are matched to compute the final GUI Score, rather than serving as an additional learned baseline.

Model	Calculator	Login Form	Music Player	Doc Editor	Image Editor	Info Card	Overall
DeepSeek-R1	27.73	**96.87**	91.73	72.32	**76.39**	**88.46**	75.58
Llama-4-Scout-17B	**73.34**	90.29	71.60	59.90	69.37	76.46	73.49
Qwen3.5-27B$^-$	70.77	96.82	**92.92**	**75.45**	75.33	86.74	**83.00**
Qwen3.5-27B$^+$	45.21	96.32	89.33	75.32	70.36	86.78	77.22
Qwen3.5-9B$^-$	54.09	95.95	88.91	75.42	73.84	86.38	79.10
Qwen3.5-9B$^+$	31.39	93.28	84.98	75.12	70.35	87.45	73.76
Qwen3.5-4B$^-$	48.83	84.91	90.63	70.16	67.42	85.83	74.63
Qwen3.5-4B$^+$	28.97	90.00	73.89	60.56	38.90	81.65	62.33

Table 2: Main evaluation results (Strict Score %) across six distinct scenarios. Superscripts $^+$ and $^-$ denote configurations with and without thinking mode, respectively. Best per column is **bold**; second-best is underlined. The Overall score is averaged over the six scenarios.

4.2 Test Scenarios and Dataset Stratification

We evaluate slot-aligned GUI generation under bilingual settings in Chinese and English to assess open-loop action planning and spatial reasoning.

Our template-based data synthesis can generate over 10,000 high-quality samples for each scenario. To ensure balanced and efficient evaluation, we sample 1,000 instances from each of the six scenarios, resulting in a total benchmark subset of 6,000 instances.

The evaluation covers six scenarios with different layout difficulties: the `calculator` focuses on dense grid alignment; the `login form` and `info card` test simple sequential layouts; the `music player` evaluates component coordination; and the `doc editor` and `image editor` challenge multi-region layout and geometric precision.Based on our schema-based data synthesis framework, each instance is programmatically generated via parameterized configurations, enabling deterministic one-to-one alignment between instruction specifications and ground-truth layouts across all evaluated models. Comprehensive scenario schemas and implementation details are provided in Appendix A.4.

4.3 Evaluation Metrics

We evaluate the models from two perspectives: **Quality Metric:** We employ the composite GUI Score (GS) and its five sub-metrics, as defined in Sec 3.2, to comprehensively assess the structural integrity and spatial precision of the generated layouts. **Efficiency Metrics:** To quantify the computational cost and complexity, we record the overall token consumption (T) for each sample, including **Prompt Tokens** (T_{in}) to measure the input overhead and **Response Tokens** (T_{out}) to evaluate the generation verbosity and reasoning depth.

5 Analysis

5.1 Performance Variance in SchemaGUI

The Qwen3.5 family demonstrates superior overall performance, outperforming strong baseline models in SchemaGUI. As shown in Table 2, Qwen3.5-27B$^-$ obtains the highest Overall score among all evaluated models, reaching 83.00%. This result clearly surpasses DeepSeek-R1 and Llama-4-Scout-17B, which achieve 75.58% and 73.49%, respectively. Notably, even the smaller Qwen3.5-9B$^-$ achieves an Overall score of 79.10%, outperforming both baselines. This suggests that the Qwen3.5 family is better aligned with the structured output requirements of SchemaGUI, where stable layout synthesis and task-specific instruction following are crucial.

Simple sequential layouts are easier for current models, while dense grids and multi-region compositions sharply increase generation difficulty. For example, DeepSeek-R1 and Qwen3.5-27B$^-$ both achieve over 96% on *Login Form* and over 91% on *Music Player*, where components are mostly arranged in a linear order. In contrast, *Calculator* contains many closely packed components and requires strict two-dimensional grid alignment, making it difficult to preserve row-column consistency without visual feedback. DeepSeek-R1 drops to 27.73% on this task, while Llama-4-Scout-17B reaches 73.34%. This sharp drop indicates that DeepSeek-R1 struggles with precise coordinate control, where small deviations in repeated button placement can break the whole grid. Ex-

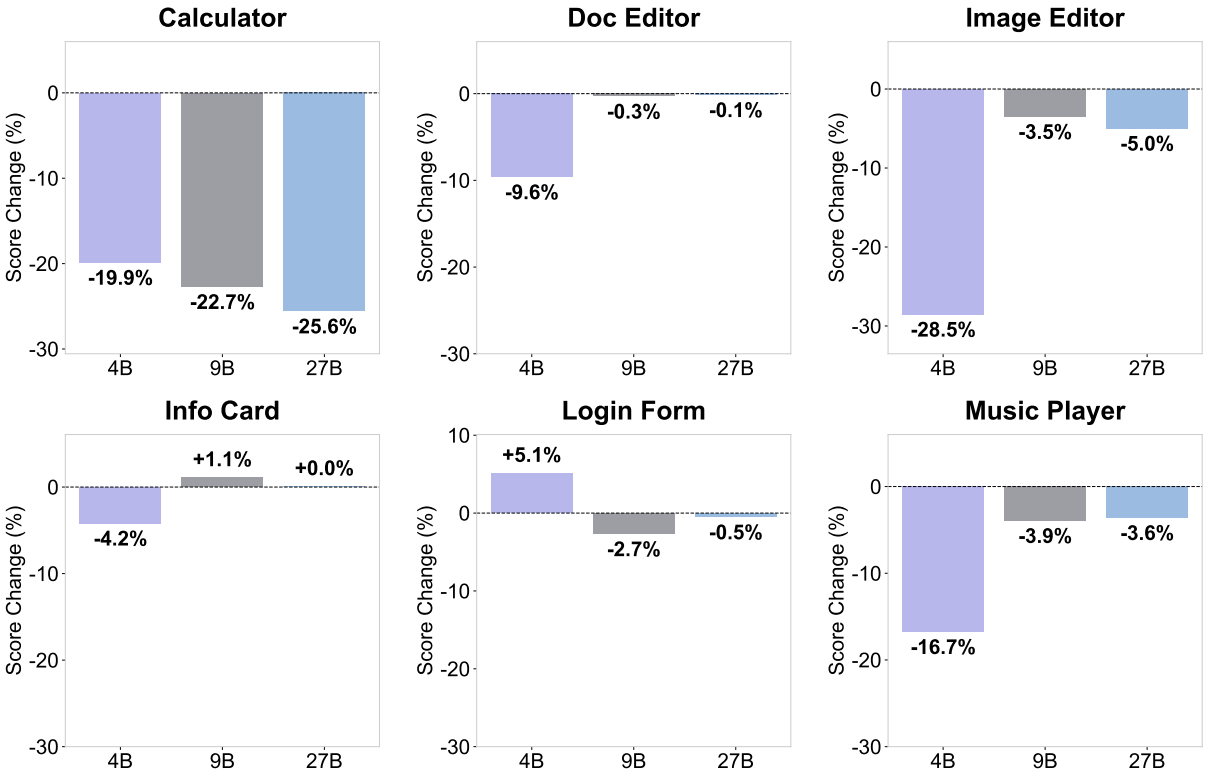

Figure 3: Performance delta between reasoning and non-reasoning configurations. Score change per scenario, defined as ΔScore = Thinking − No Thinking, where positive values indicate improvement, while negative values indicate performance loss.

cessive reasoning may further introduce redundant content, disrupting the function-call structure and causing layout failure. For the multi-region *Image Editor* task, models must coordinate regions such as the toolbar, sidebar, and canvas. DeepSeek-R1 achieves 76.39%, while Qwen3.5-4B^{-} reaches 67.42%. These scenario gaps suggest that geometry-sensitive layouts require more than semantic instruction understanding; we further examine this limitation through submetric and feasibility analysis in Section 5.4.

5.2 The Impact of Reasoning

Long-form reasoning can amplify geometric layout errors under strict GUI generation constraints. As illustrated by the performance deltas in Figure 3, the effect of the thinking mode varies significantly across layout structures. In coordinate-dense tasks such as `calculator`, enabling reasoning clearly impairs layout precision, with the score dropping by 19.86% for Qwen3.5-4B and by 25.56% for Qwen3.5-27B. A similar pattern appears in `image editor`, where the 4B variant drops sharply from 67.42% to 38.90%. In contrast, simple sequential layouts such as `login form` and `info card` remain relatively stable, with score changes clustering around the zero baseline.

This suggests that the bottleneck in GUI generation is not simply reasoning depth, but precise geometric execution under strict schema constraints. While long-form reasoning may help with step-by-step logical tasks, GUI layout generation requires strict instruction following, valid function-call formatting, and precise spatial planning. In coordinate-heavy scenarios, excessive reasoning may introduce redundant intermediate content, disturb the required output structure, and increase the risk of coordinate drift or layout inconsistency.

5.3 Token Consumption in SchemaGUI

Response-token consumption increases with the structural complexity of GUI tasks.

Table 3 shows that the main variation appears in T_{out}, which captures the response-side cost of generating GUI function-call sequences. From the scenario averages, `calculator` requires the largest response length, with an average of 612.6 tokens, followed by `image editor` with 441.8 tokens. In contrast, simpler layouts such as `login form` and `info card` require much lower re-

Model (Input T_{in})	Calculator (743.9)	Login Form (495.7)	Music Player (503.2)	Doc Editor (529.5)	Image Editor (587.0)	Info Card (605.3)	Overall (Avg. T_{out})
DeepSeek-R1	408.5	174.8	264.2	287.6	376.6	190.2	283.7
Llama-4	622.6	**251.5**	303.5	285.1	399.4	**246.2**	351.4
Qwen3.5-27B$^-$	625.9	172.2	264.2	257.2	361.5	168.0	308.2
Qwen3.5-27B$^+$	586.4	206.7	324.9	312.5	451.9	224.2	351.1
Qwen3.5-9B$^-$	**821.2**	226.8	<u>373.6</u>	<u>361.7</u>	<u>500.0</u>	<u>235.0</u>	**419.7**
Qwen3.5-9B$^+$	568.8	229.1	353.5	342.2	460.3	230.4	364.1
Qwen3.5-4B$^-$	<u>753.2</u>	<u>243.7</u>	**375.5**	**366.4**	**508.8**	232.5	<u>413.3</u>
Qwen3.5-4B$^+$	513.8	239.7	319.6	304.8	476.0	221.3	345.9
Average	**612.6**	218.1	322.4	314.7	<u>441.8</u>	218.5	—

Table 3: Token usage metrics across six GUI scenarios. The final column reports average T_{out} per model, and the final row reports average T_{out} per scenario. Token statistics are normalized under the same counting protocol to support cross-model comparison.Bold and underlined values mark the highest and second-highest output-token costs.

sponse costs, with averages of 218.1 and 218.5 tokens, respectively. This shows that component-dense grids and multi-region layouts require more function calls, component attributes, and geometric parameters, leading to higher token consumption.

Comparing Table 2 and Table 3, Qwen3.5-27B$^-$ obtains the highest Overall score of 83.00% with an average of 308.2 output tokens. This is much lower than Qwen3.5-9B$^-$ and Qwen3.5-4B$^-$, which use 419.7 and 413.3 tokens but achieve lower Overall scores. Although DeepSeek-R1 produces shorter responses, its lower Overall score and severe degradation on `calculator` show that fewer tokens do not necessarily indicate better generation quality. Effective GUI generation therefore requires a balance between compact function-call generation, structural coverage, and accurate layout control.

5.4 Performance Across UI Complexities

Model	$\mathcal{G}_{\text{ok}}$	$\mathcal{F}_{\text{ok}}$	$\mathcal{P}_{\text{ok}}$	$\mathcal{O}_{\text{ok}}$	$\mathcal{S}_{\text{ok}}$	Feasibility
Qwen3.5-4B	75.47	83.55	84.46	93.35	100.00	89.33
Qwen3.5-9B	76.07	82.12	84.14	91.32	100.00	95.23
Qwen3.5-27B	**80.04**	**85.02**	**84.95**	**95.69**	100.00	**96.98**
DeepSeek-R1	76.30	79.97	79.54	90.63	100.00	81.20
Llama-4	73.88	73.40	73.11	95.62	100.00	78.20

Table 4: Detailed performance (%) and structural feasibility across models, showing averaged strict submetrics for valid samples.The subscript ok denotes submetric scores computed only on schema-feasible outputs.

Current LLMs still struggle with precise geometric layout control. Table 4 demonstrates that schema validity and text-level compliance do not fully determine GUI quality; precise geometric layout control remains the limiting factor. Within the Qwen3.5 family, scaling from 4B to 27B increases structural feasibility from 89.33% to 96.98%, showing that larger models are better at producing schema-valid function-call outputs. However, the precise geometric alignment score $\mathcal{G}_{\text{ok}}$ only improves from 75.47% to 80.04%, a gain of 4.57 percentage points. Meanwhile, textual compliance $\mathcal{P}_{\text{ok}}$ stays relatively stable around 84.5%, indicating that text matching is not the main source of variation across Qwen scales.

The cross-model comparison further confirms this limitation. DeepSeek-R1 and Llama-4 achieve only 76.30% and 73.88% on $\mathcal{G}_{\text{ok}}$, both below Qwen3.5-27B. Although Llama-4 reaches 95.62% on $\mathcal{O}_{\text{ok}}$, its geometry score is the lowest, suggesting that preserving call order is not enough for high-quality GUI generation. Accurate coordinate and size control remains the key challenge.

6 Conclusion

We introduce **SchemaGUI**, a template-based benchmark for controllable GUI generation evaluation. It produces paired natural-language instructions and deterministic function-call references across six GUI scenarios. Experiments show that larger models improve schema compliance and overall performance, but geometric layout control remains the main bottleneck. Geometry errors remain harder to resolve than text or structure mismatches, and explicit reasoning can hurt coordinate-dense layouts. These findings highlight the need for geometry-aware, schema-constrained evaluation.

Limitations

Despite its robustness, SchemaGUI has several limitations. First, it focuses on static structural synthesis and geometric layout planning, leaving dynamic user interactions and sequential runtime workflows unexplored. Second, the current evaluation follows a text-to-schema setting, while practical GUI design often depends on multimodal feedback loops that combine textual intent, visual inspection, and iterative refinement. Third, although the template-based design enables controllable and deterministic evaluation, it may not fully capture the visual diversity, stylistic variation, and open-ended constraints of real-world interfaces. In addition, our metrics mainly emphasize structural correctness and geometric fidelity, while subjective visual factors such as aesthetics and design preference are not explicitly evaluated. Finally, although our experiments cover representative mainstream LLMs, future evaluation on more model families and larger sample sets may provide a more comprehensive understanding of model behavior.Future work will extend SchemaGUI toward interactive environments, richer GUI scenarios, human-centered visual assessment, and vision-language feedback for iterative layout self-correction.

Ethical Concern

This work focuses on the construction and evaluation of a controlled benchmark for GUI generation. All samples in SchemaGUI are programmatically generated from predefined schemas and parameterized templates. The benchmark does not rely on real user interfaces, private screenshots, personal identifiers, user behavior logs, or any sensitive personal data. All annotations are automatically derived from deterministic templates, and the experiments are conducted in offline evaluation environments. Therefore, the current benchmark does not involve direct interaction with users or real-world decision-making processes.

The main ethical risks of this work are indirect. SchemaGUI is designed to evaluate structural correctness, geometric layout quality, and schema-following ability in GUI generation models. It should not be interpreted as a guarantee that a generated interface is safe, accessible, visually appropriate, or ready for deployment in user-facing systems. If similar GUI generation techniques are later used in practical design workflows, human review and additional validation would be necessary to prevent misleading layouts, inaccessible designs, or interfaces that fail to satisfy real user needs.

We also acknowledge that template-based generation provides strong controllability but may simplify the open-ended nature of real GUI design. As a result, the benchmark is intended as a diagnostic tool for measuring layout and structural generation ability, rather than a replacement for human-centered design evaluation. Future extensions involving real-world interfaces, interactive environments, or user-facing deployment should include stronger safeguards, such as privacy-aware data handling, accessibility checks, human evaluation, and careful review of generated interface behavior.

During the research process, AI tools may be used to support auxiliary tasks such as code debugging, writing refinement, and visualization preparation. All core research ideas, benchmark design choices, experimental analysis, and final conclusions are reviewed and determined by the authors. The use of AI assistance does not replace human responsibility for the correctness, validity, and ethical treatment of the research.

References

Aaron Adcock, Aayushi Srivastava, Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pande, Abhinav Pandey, Abhinav Sharma, Abhishek Kadian, Abhishek Kumawat, Adam Kelsey, and 1 others. 2026. The llama 4 herd: Architecture, training, evaluation, and deployment notes. *arXiv preprint arXiv:2601.11659*.

Tony Beltramelli. 2017. pix2code: Generating code from a graphical user interface screenshot. *Preprint*, arXiv:1705.07962.

Yongxi Chen and Lei Chen. 2025. Psd2code: Automated front-end code generation from design files via multimodal large language models. *Preprint*, arXiv:2511.04012.

Yunnong Chen, Chengwei Shi, and Liuqing Chen. 2025. Specifyui: Supporting iterative ui design intent expression through structured specifications and generative ai. *Preprint*, arXiv:2509.07334.

Peitong Duan, Chin-Yi Cheng, Gang Li, Bjoern Hartmann, and Yang Li. 2024. Uicrit: Enhancing automated design evaluation with a ui critique dataset. In *Proceedings of the 37th Annual ACM Symposium on User Interface Software and Technology*, UIST '24, New York, NY, USA. Association for Computing Machinery.

Weixi Feng, Wanrong Zhu, Tsu jui Fu, Varun Jampani, Arjun Akula, Xuehai He, Sugato Basu, Xin Eric

Wang, and William Yang Wang. 2023. Layoutgpt: Compositional visual planning and generation with large language models. *Preprint*, arXiv:2305.15393.

Zhouhong Gu, Haoning Ye, Xingzhou Chen, Zeyang Zhou, Hongwei Feng, and Yanghua Xiao. 2024. Structext-eval: Evaluating large language model's reasoning ability in structure-rich text. *Preprint*, arXiv:2406.10621.

Yi Gui, Yao Wan, Zhen Li, Zhongyi Zhang, Dongping Chen, Hongyu Zhang, Yi Su, Bohua Chen, Xing Zhou, Wenbin Jiang, and Xiangliang Zhang. 2025. Uicopilot: Automating ui synthesis via hierarchical code generation from webpage designs. In *Proceedings of the ACM on Web Conference 2025*, WWW '25, page 1846–1855, New York, NY, USA. Association for Computing Machinery.

Daya Guo, Dejian Yang, Haowei Zhang, Junxiao Song, Peiyi Wang, Qihao Zhu, Runxin Xu, Ruoyu Zhang, Shirong Ma, Xiao Bi, Xiaokang Zhang, Xingkai Yu, Yu Wu, Z. F. Wu, Zhibin Gou, Zhihong Shao, Zhuoshu Li, Ziyi Gao, Aixin Liu, and 175 others. 2025. Deepseek-r1 incentivizes reasoning in llms through reinforcement learning. *Nature*, 645(8081):633–638.

Daeheon Jeong, Seoyeon Byun, Kihoon Son, Dae Hyun Kim, and Juho Kim. 2025. Canvas: A benchmark for vision-language models on tool-based user interface design. *Preprint*, arXiv:2511.20737.

Yilei Jiang, Yaozhi Zheng, Yuxuan Wan, Jiaming Han, Qunzhong Wang, Michael R. Lyu, and Xiangyu Yue. 2025. Screencoder: Advancing visual-to-code generation for front-end automation via modular multimodal agents. *Preprint*, arXiv:2507.22827.

Akash Abdu Jyothi, Thibaut Durand, Jiawei He, Leonid Sigal, and Greg Mori. 2021. Layoutvae: Stochastic scene layout generation from a label set. *Preprint*, arXiv:1907.10719.

Kristian Kolthoff, Felix Kretzer, Christian Bartelt, Alexander Maedche, and Simone Paolo Ponzetto. 2025. Guide: Llm-driven gui generation decomposition for automated prototyping. *Preprint*, arXiv:2502.21068.

Yaniv Leviathan, Dani Valevski, Matan Kalman, Danny Lumen, Eyal Segalis, Eyal Molad, Shlomi Pasternak, Vishnu Natchu, Valerie Nygaard, Srinivasan, Venkatachary, James Manyika, and Yossi Matias. 2026. Generative ui: Llms are effective ui generators. *Preprint*, arXiv:2604.09577.

Liulei Li, Tianfei Zhou, Wenguan Wang, Jianwu Li, and Yi Yang. 2022. Deep hierarchical semantic segmentation. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 1246–1257.

Shanchao Liang, Nan Jiang, Shangshu Qian, and Lin Tan. 2026. Waffle: Finetuning multi-modal models for automated front-end development. *Preprint*, arXiv:2410.18362.

Yimeng Liu, Misha Sra, and Chang Xiao. 2026. Alignui: A method for designing llm-generated uis aligned with user preferences. *Preprint*, arXiv:2601.17614.

Yuwen Lu, Ziang Tong, Qinyi Zhao, Chengzhi Zhang, and Toby Jia-Jun Li. 2023. Ui layout generation with llms guided by ui grammar. *Preprint*, arXiv:2310.15455.

Qwen Team. 2026. Qwen3.5: Towards native multimodal agents.

Alex Robinson. 2019. Sketch2code: Generating a website from a paper mockup. *Preprint*, arXiv:1905.13750.

Chenglei Si, Yanzhe Zhang, Ryan Li, Zhengyuan Yang, Ruibo Liu, and Diyi Yang. 2024. Design2code: Benchmarking multimodal code generation for automated front-end engineering. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 7406–7427. Association for Computational Linguistics.

Liangtai Sun, Xingyu Chen, Lu Chen, Tianle Dai, Zichen Zhu, and Kai Yu. 2022. Meta-gui: Towards multi-modal conversational agents on mobile gui. *Preprint*, arXiv:2205.11029.

Hao Yang, Weijie Qiu, Ru Zhang, Zhou Fang, Ruichao Mao, Xiaoyu Lin, Maji Huang, Zhaosong Huang, Teng Guo, Shuoyang Liu, and Hai Rao. 2025. Ui-ug: A unified mllm for ui understanding and generation. *Preprint*, arXiv:2509.24361.

Houston H. Zhang, Tao Zhang, Baoze Lin, Yuanqi Xue, Yincheng Zhu, Huan Liu, Li Gu, Linfeng Ye, Ziqiang Wang, Xinxin Zuo, Yang Wang, Yuanhao Yu, and Zhixiang Chi. 2026. Widget2code: From visual widgets to ui code via multimodal llms. *Preprint*, arXiv:2512.19918.

Shuyan Zhou, Frank F. Xu, Hao Zhu, Xuhui Zhou, Robert Lo, Abishek Sridhar, Xianyi Cheng, Tianyue Ou, Yonatan Bisk, Daniel Fried, Uri Alon, and Graham Neubig. 2024. Webarena: A realistic web environment for building autonomous agents. *Preprint*, arXiv:2307.13854.

A Appendix

A.1 Weighting Schemes and Ranking Consistency

We define the Primary scheme as the default weighting setting for computing the GUI Score. It assigns a relatively higher weight to geometry because coordinate and size accuracy are central to visual correctness and usability in GUI layouts. To examine whether our conclusions depend on this weighting choice, we compare it with several alternative schemes that redistribute the weights across the five submetrics.

Table 5 reports the exact weight vectors $\mathbf{w} = (w_{\mathcal{G}}, w_{\mathcal{F}}, w_{\mathcal{P}}, w_{\mathcal{O}}, w_{\mathcal{S}})$ used in the sensitivity analysis.

Scheme	$w_{\mathcal{G}}$	$w_{\mathcal{F}}$	$w_{\mathcal{P}}$	$w_{\mathcal{O}}$	$w_{\mathcal{S}}$
Primary	0.35	0.25	0.20	0.15	0.05
Alt-A (Str.-shifted)	0.30	0.25	0.20	0.15	0.10
Uniform	0.20	0.20	0.20	0.20	0.20
Geometry-emphasized	0.45	0.20	0.15	0.10	0.10
Structure-emphasized	0.25	0.20	0.15	0.15	0.25

Table 5: Exact weight vectors used in the sensitivity analysis.

Table 6 summarizes the main weight-sensitivity results on shared scenarios. Changing the weighting scheme affects the absolute GS values but does not change the model ranking. For example, Qwen3.5-27B increases from 83.62% under Primary to 86.58% under Uniform and 86.93% under Structure-emphasized. Similar increases are also observed for Qwen3.5-4B and Qwen3.5-9B. However, the ranking remains unchanged across all three schemes: Qwen3.5-27B performs best, followed by Qwen3.5-9B and Qwen3.5-4B. This indicates that the main benchmark conclusions are stable under different scoring priorities.

Model	Scheme	GS (%)
Qwen3.5-4B	Primary	68.67
	Uniform	71.63
	Structure-emphasized	71.73
Qwen3.5-9B	Primary	82.23
	Uniform	85.63
	Structure-emphasized	85.90
Qwen3.5-27B	Primary	83.62
	Uniform	86.58
	Structure-emphasized	86.93

Table 6: Main weight-sensitivity results on shared scenarios. Scores are reported as $\mathrm{GS}_{100} = 100 \times \mathrm{GS}$.

A.2 Detailed Scenario Performance Deltas

Model	Calc.	Login	Music	Doc.	Image	Info.
Qwen3.5-27B	-25.56	-0.50	-3.59	-0.13	-4.97	+0.04
Qwen3.5-9B	-22.70	-2.67	-3.93	-0.30	-3.49	+1.07
Qwen3.5-4B	-19.86	+5.09	-16.74	-9.60	-30.16	-4.18

Table 7: Absolute performance change ($\Delta = \mathrm{Score}_{+} - \mathrm{Score}_{-}$) across six GUI scenarios after activating the thinking mode. Negative values indicate lower performance under the reasoning-enabled setting.

Table 7 provides the exact numerical deltas corresponding to the reasoning analysis in Section 5.2. The largest drops are concentrated in coordinate-sensitive scenarios such as *Calculator* and *Image Editor*, while simpler layouts such as *Login Form* and *Info Card* show comparatively smaller changes.

A.3 Token Range Analysis

Table 8 complements the average token statistics in the main text by showing the empirical range of token consumption. The input-token ranges remain narrow and nearly identical within each scenario, confirming that the prompt side is controlled by the SchemaGUI generation process. In contrast, response-token ranges vary substantially, especially in the *Calculator* scenario. For example, Qwen3.5-9B^{-} reaches a maximum of 3035 response tokens, while several reasoning-enabled variants have a minimum of only 47 tokens. This wide range suggests that coordinate-dense layouts are more likely to trigger unstable generation behavior, from overly verbose outputs to prematurely shortened or incomplete function-call sequences. By comparison, simpler scenarios such as *Login Form* and *Info Card* show narrower response ranges, indicating more stable generation patterns.

A.4 Scenario Schemas and Implementation Details

Listing 1 presents the general prompt template used for schema-driven GUI generation. For each sampled instance state s, the placeholders in the template, such as `{scenario}`, `{window_size}`, `{layout}`, `{component_list}`, and `{scenario_guidance}`, are filled with the corresponding configuration values defined by s. The completed prompt is then provided to the LLM as the input instruction x, from which the model generates the predicted function-call sequence y^L. The template combines scenario information, allowed functions, required components, and layout guidance into a unified prompt, while enforcing a strict JSON-only function-call output format.

To adapt the same template to different UI scenarios, the field `{scenario_guidance}` injects scenario-specific layout constraints. For example, *Calculator* emphasizes compact grid grouping, *Login Form* and *Info Card* emphasize sequential structure and readability, *Music Player* focuses on playback-oriented control grouping, and *Doc Editor* and *Image Editor* require clear multi-region separation.

Model	Type	Calculator	Login Form	Music Player	Doc Editor	Image Editor	Info Card
DeepSeek-R1	T_{in}	[713–773]	[466–511]	[482–525]	[510–556]	[555–624]	[574–623]
	T_{out}	[49–726]	[110–235]	[203–325]	[195–449]	[279–632]	[106–263]
Llama-4	T_{in}	[713–773]	[466–511]	[482–525]	[510–556]	[549–629]	[574–625]
	T_{out}	[541–755]	[154–364]	[232–405]	[222–382]	[275–538]	[152–316]
Qwen3.5-27B$^-$	T_{in}	[713–773]	[466–511]	[482–525]	[510–556]	[549–629]	[574–625]
	T_{out}	[522–1421]	[110–235]	[203–325]	[196–342]	[271–452]	[106–194]
Qwen3.5-27B$^+$	T_{in}	[713–773]	[466–511]	[482–525]	[510–556]	[549–629]	[574–625]
	T_{out}	[47–1006]	[110–434]	[203–463]	[196–492]	[270–629]	[107–336]
Qwen3.5-9B$^-$	T_{in}	[713–773]	[466–511]	[482–525]	[510–556]	[549–629]	[574–625]
	T_{out}	[217–3035]	[133–331]	[286–460]	[278–488]	[311–624]	[150–451]
Qwen3.5-9B$^+$	T_{in}	[713–773]	[466–511]	[482–525]	[510–556]	[549–629]	[574–625]
	T_{out}	[47–2173]	[110–334]	[203–463]	[196–493]	[279–629]	[107–267]
Qwen3.5-4B$^-$	T_{in}	[713–773]	[466–511]	[482–525]	[510–556]	[549–629]	[574–625]
	T_{out}	[524–2135]	[154–331]	[286–583]	[278–487]	[378–686]	[150–264]
Qwen3.5-4B$^+$	T_{in}	[713–773]	[466–511]	[482–525]	[510–556]	[549–629]	[574–625]
	T_{out}	[47–759]	[111–434]	[203–488]	[196–587]	[348–625]	[107–339]

Table 8: Inference token ranges across six GUI scenarios. Each cell reports the empirical minimum and maximum token counts $[\min - - \max]$ for T_{in} and T_{out}. Superscripts $^+$ and $^-$ denote configurations with and without thinking mode, respectively.

```
PROMPT_TEMPLATE = """You must output a
    pure JSON array. Do not include any
    explanations, markdown, thinking
    processes, or ellipses.

Scenario: {scenario}
Window: {window_size}
Layout: {layout}

Allowed functions: {functions}
Output format: [{"name":...,"parameters
    ":...}, ...]

Hard rules:
 - The first call must be set_window.
 - All add_* calls must include integer
     x,y,width,height and a string text.

Component List (must include all,
    generate one by one):
{component_list}

Base rules for every scenario:
Group related elements into compact
    regions and keep different
    functional areas visually separated.
Maintain consistent spacing and row/
    column alignment where applicable.

Scenario-specific guidance for each
    scenario:
{scenario_guidance}

Now, output only the JSON array and
    nothing else:
"""
```

Listing 1: Prompt template for schema-driven GUI generation.

Prompt and reference construction. Each evaluation instance is derived from a shared scenario-specific instance state s, which contains the sampled interface configuration, including window size, layout style, component inventory, textual content, and visual constraints. SchemaGUI uses this same s in two parallel branches. In the prompt branch, s is verbalized into a natural-language instruction and inserted into the general template in Listing 1, producing the model input x. The LLM then generates the predicted output y^L from this textual prompt. In the reference branch, the same s is deterministically expanded into the canonical function-call sequence y^T, which is hidden from the model and used only as the evaluation reference.

This design ensures that the prompt x and reference y^T are aligned by construction, while still allowing the model to generate its own layout prediction y^L from the textual instruction. For example, a *Calculator* instance may specify a 720×640 window, a standard grid layout, a dual-line display, and button groups including numeric keys, arithmetic operators, and functional keys such as AC and =. The prompt branch describes these requirements in natural language, whereas the reference branch generates the corresponding canonical function-call layout. During evaluation, the predicted sequence y^L is parsed and compared with y^T to assess schema compliance.