CANDO: Cooperative Agentic Network for Layout Design Optimization

Athanasios Masouris[1] **Jing Zheng**[2] **Benjamin Chandler**[1] **Hadi Jamali-Rad**[1,3]

[1] Shell Information Technology International, [2] Shell China Limited
[3] Delft University of Technology (TU Delft)

Abstract

Layout generation for real-world facilities is a challenging problem, requiring reasoning over irregular site boundaries, heterogeneous orientations, access-aware placements, and motion-planning feasibility. Yet, most existing layout benchmarks in the generative AI space target simpler placements over rectangular domains and rely on distributional metrics such as FID and IoU that reward conformity to dataset priors, thus discounting design innovation. Motivated by these gaps, we introduce **ALPS-Bench**, a benchmark of $1,000$ professionally annotated real-world facility layouts paired with an instance-specific scoring protocol grounded in a structured design manual. As a strong baseline for **ALPS-Bench**, we propose `CANDO`, a training-free multi-agent framework in which specialized agents iteratively refine layouts through a verification-grounded loop, concentrating reasoning on strategic spatial decisions. We demonstrate that `CANDO` surpasses state-of-the-art trained and LLM-based baselines on the widely adopted **PubLayNet**, **RICO**, and **PKU-PosterLayout** benchmarks, establishing cooperative agentic design as a broadly effective recipe for constraint-aware layout synthesis.

Project Page **Codebase** **ALPS-Bench**

1 Introduction

Layout generation has recently received an upsurge of attention as a subfield of generative modeling, with strong results reported across floor planning [54, 24, 49, 1], document design [59, 39], poster design [17], and graphical user interfaces synthesis [7, 52] tasks. Yet the regime captured by these interesting benchmarks is notably simpler than real-world design practice: floor-planning datasets are dominated by indoor, largely axis-aligned room arrangements; document and web layouts reduce to rectangular composition over discrete canvases. Besides, their evaluation strategy is predominantly distributional, with metrics such as FID, IoU, SSIM, and graph similarity rewarding resemblance to the dataset prior. The consequence is twofold. First, design innovation is implicitly penalized; i.e., valid yet atypical configurations are scored down simply for departing from the empirical distribution, collapsing the task into prior matching. Second, methods that excel under these protocols rarely transfer to settings where compositionality, orientation, and inter-element spatial relations are first-class concerns rather than emergent regularities of the dataset.

A canonical real-world instance of this overlooked regime is asset layout design: arranging service units, parking bays, access points, and auxiliary buildings within an operational facility. Unlike its document and floor-plan counterparts, this problem requires reasoning over irregular, instance-specific site boundaries; enforcing orientation consistency across heterogeneous functional groups; organizing elements with respect to designated access points; and certifying motion-planning feasibility for multiple vehicle scenarios traversing the site. These requirements are entangled, namely: a placement that satisfies non-overlap may still block a routing scenario, and a compositionally clean clustering may violate access-aware constraints. We find empirically that neither distributional metrics nor (well-equipped) single-agent baselines can adequately address this challenging problem. More specifically,

40th Conference on Neural Information Processing Systems (NeurIPS 2026).

monolithic generators struggle on precisely the joint demands of spatial arithmetic, multi-constraint reasoning, and long-horizon planning that the problem entails, pointing to the need for an explicitly decomposed, verification-grounded solution architecture. Intrigued by these gaps and challenges, we make the following three core contributions:

I. We introduce the Asset Layout Planning and Synthesis Benchmark (**ALPS-Bench**): 1,000 real-asset professionally-annotated layouts, each paired with a structured design manual of diverse spatial constraints. Our proposed evaluation protocol scores compliance across a variety of design criteria for standardized evaluation on a held-out test set.

II. We propose CANDO, to the best of our knowledge, one of the first cooperative agentic AI frameworks for constraint-aware layout synthesis and a strong baseline for **ALPS-Bench**. CANDO offers several innovative design choices, including structured cooperation among agents, tool-grounded verification, and motion planning.

III. We demonstrate that even a simplified version of CANDO achieves state-of-the-art mIoU and alignment on the widely adopted **PubLayNet** and **RICO** benchmarks, and leading overlap and strict-underlay performance on **PKU-PosterLayout**, corroborating that cooperative agentic frameworks are a broadly effective recipe for layout generation.

2 Related Work

Layout Design and Its Benchmarks. Automated layout generation has been studied across a range of domains, including graphical user interfaces [7], document design [59], content-aware poster design [17], indoor scene arrangement [38, 47], and architectural floor planning [35, 43]. Early work relied on GANs [31] and VAEs [23, 29], achieving reasonable qualitative results but at the cost of task-specific training on large labeled datasets. Transformer-based methods then broadened the scope, with LayoutTransformer [14] enabling layout completion via self-attention, and LayoutFormer++ [22] unifying diverse constraint types into a single sequence format. Diffusion models followed, with LayoutDM [19], LayoutFlow [13], and DiffuScene [47] pushing generation quality further. These methods require training and are evaluated on distributional metrics like FID and overlap that reward proximity to the training prior rather than actual constraint satisfaction. Large language models (LLMs) have offered a different path. LayoutPrompter [32] and LayoutGPT [10] pioneered training-free layout generation via in-context learning (ICL), while fine-tuned variants such as LayoutNUWA [48] and PosterLlama [42] improved quality by representing layouts as structured HTML code. More recently, LayoutCoT [44] incorporated chain-of-thought (CoT) reasoning, while LaRa [27] introduced reinforcement learning (RL) from verifiable rewards. An emerging agentic direction pushes this further, with DisCo-Layout [11] and AutoLayout [4] proposing multi-agent loops for iterative spatial refinement. Notably, RICO [7] and PubLayNet [59] remain the predominant benchmarks adopted by almost all of the above, focusing on much simpler tasks and relying on prior matching metrics (such as mIoU and FID), leaving a gap in the literature that jointly motivated the proposed **ALPS-Bench** and CANDO.

Cooperative Agentic AI. The structured reasoning capabilities of LLMs through CoT have been further enhanced by methods that guide such models to verify and refine their own responses, including Self-Refine [34] and Reflexion [45]. ReAct [58] established the interplay between structured reasoning and environment-grounded actions, paving the way for the modern agentic systems. Toolformer [41] demonstrated that LLMs benefit from offloading computations to external tools, which is particularly important given their known spatial arithmetic limitations [3]. Frameworks, such as Generative Agents [37], MemGPT [36], and A-Mem [55], further showed that episodic memory management is essential for coherent long-horizon behavior. Building upon these findings, multi-agent orchestration frameworks, such as AutoGen [53], MetaGPT [15], CAMEL [30], AgentVerse [5], and MINDcraft [51], establish that role-specialized cooperating agents consistently outperform monolithic models on complex tasks [8]. Inspired by this prior art, we propose CANDO, a novel approach to constraint-aware layout design consisting of a cooperative agentic network that leverages tool-grounded verification, episodic feedback memory, role-specialized cooperation, and block-wise BoN [2] (a.k.a CoDe [46]) into a single training-free framework.

3 Problem Formulation

We formalize layout synthesis as constrained optimization of rigid-body placement over a 2D plane. The goal is to determine the position and orientation of each element so as to satisfy geometric feasibility requirements and minimize violation of design constraints.

Layout Domain. It is comprised of the layout premise $\mathcal{P} \subset \mathbb{R}^2$; an empty outer polygon within which the elements are to be placed. And, a set of access points (i.e., ENTRANCE, EXIT) sitting on the edges of the premise $\mathcal{P}$. Access points define designated regions through which vehicles may enter or leave the domain, and they take a pivotal role in operational constraints such as routing feasibility.

Elements Configuration. Elements e_i ($i \in [E]$) are to be placed within the premise $\mathcal{P}$. Each $e_i = (c_i, G_i)$ has a semantic class $c_i \in \mathcal{C}$ and a fixed canonical shape/body $G_i \subset \mathbb{R}^2$, a compact region centered at the origin (e.g. a rectangle). We denote $\mathcal{E} = \{(c_i,\, G_i)\}_{i=1}^{E}$ for the full element specification. Element shapes/sizes, classes, and counts are *fixed inputs*; only their spatial placement is optimized. See Fig. 1 for an example layout illustrating premise, access points and elements.

Each e_i is displaced by assigning a centroid $\mathbf{p}_i = (x_i, y_i) \in \mathbb{R}^2$ and an orientation angle $\theta_i \in [0, 2\pi)$. The displaced element is the element canonical shape G_i rotated by θ_i and translated to $\mathbf{p}_i$ following:

$$\Omega_i(\mathbf{p}_i, \theta_i) = R(\theta_i)\, G_i + \mathbf{p}_i, \qquad R(\theta_i) = \begin{pmatrix} \cos\theta_i & -\sin\theta_i \\ \sin\theta_i & \cos\theta_i \end{pmatrix}. \tag{1}$$

For the sake of simplicity, we omit the fixed canonical shape G_i and class label c_i in denoting the *displaced elements* Ω_i resorting only to the parameters $(\mathbf{p}_i, \theta_i)$. We bundle these tuples in the *elements configuration* $\mathbf{q} = \{(\mathbf{p}_i, \theta_i)\}_{i=1}^{E} \in \mathcal{Q}$, with $\mathcal{Q}$ being the set of all possible configurations. As a result, $\mathbf{q}$ now can serve as our optimization parameter, focusing only on the variable parts of the elements and not the layout domain (premise and access points).

Constraints. Design manual $\mathcal{M} = \{\mathcal{HC}, \mathcal{SC}\}$ covers two sets of desired specifications:

Hard constraints ($\mathcal{HC}$) encode three necessary conditions that yield *valid* configurations $\mathcal{Q}_{\text{valid}} \subset \mathcal{Q}$:

(i) **Containment.** Every placed element lies within the premise: $\Omega_i \subseteq \mathcal{P},\ \forall i \in [E]$.
(ii) **Non-overlap.** Distinct elements do not intersect or fully overlap: $\Omega_i \cap \Omega_j = \emptyset, \forall i \neq j$.
(iii) **Routing feasibility.** A finite set of predefined scenarios $\mathcal{R}$ specifies required vehicle routes as ordered sequences of intermediate points (e.g., [VEHICLE TYPE, ENTRANCE $\rightarrow$ WAYPOINT $\rightarrow$ $\cdots$ $\rightarrow$ EXIT]). For every scenario and every concrete instance of intermediate points, a collision-free path must exist through the free space $\mathcal{P} \setminus \bigcup_i \Omega_i$.

Soft constraints ($\mathcal{SC}$) quantify how well a configuration satisfies the following design properties:

(i) **Compositionality.** Prescribes arrangement patterns and intra-group distance thresholds.
(ii) **Orientation.** Defines permitted orientations and intra-group element-wise alignment.
(iii) **Proximity.** Encodes stacking relations, clearance zones, and inter-element distance bounds.

Objective. The goal is to find a valid configuration that maximizes total weighted score:

$$\boxed{\mathbf{q}^\star = \underset{\mathbf{q} \in \mathcal{Q}_{\text{valid}}}{\arg\max}\ \mathcal{J}(\mathbf{q}) \ \triangleq\ \sum_{s=1}^{S} w_s\, f_s(\mathbf{q}),} \tag{2}$$

with $S = |\mathcal{SC}|$, and scoring function $f_s \in [0, 1]$ indicating compliance against soft constraint $s \in \mathcal{SC}$ when $f_s(\mathbf{q}) = 1$. Scoring weights sum up to $\sum_s w_s = 1$. A configuration is considered *compliant* if it is valid (satisfying all hard constraints $\mathcal{HC}$) and $\mathcal{J}(\mathbf{q}) = 1$.

Definition 1 (Input requirements)**.** *A layout input requirement is a tuple $\mathcal{I} = (\mathcal{P},\ \mathcal{A},\ \mathcal{E}, \mathcal{M})$, specifying the layout premise, access points, element set, and constraint partition.*

Remark 1 (Generalization to higher dimensions)**.** *The formulation extends naturally to volumetric layout synthesis (e.g., multi-floor facility planning) by extending Eq. 1 to its 3-D counterpart [12].*

4 Asset Layout Planning and Synthesis (ALPS) Benchmark

As discussed in Section 1, prior layout benchmarks do not fully capture the geometric and operational intricacies of real-life complex design problems such as outdoor facility/asset design, and are typically

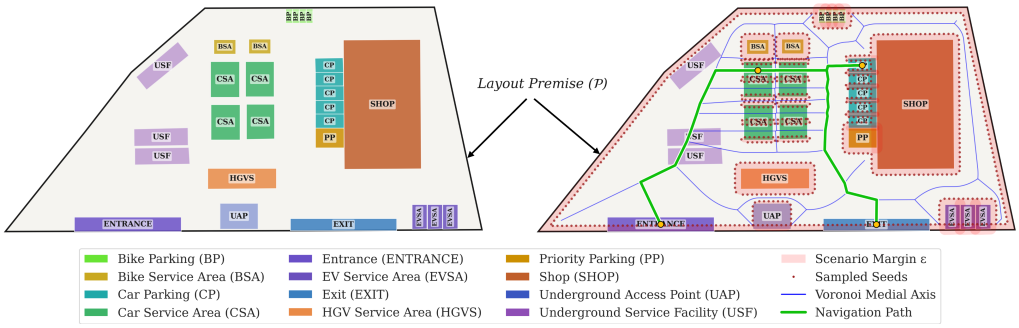

Figure 1: **Left**: Sample from **ALPS-Bench**. **Right**: Voronoi-based motion planning.

evaluated by similarity scores against dataset priors rather than constraint satisfaction. Intrigued by these gaps, we introduce the Asset Layout Planning and Synthesis Benchmark (**ALPS-Bench**), a collection of 1,000 annotated layouts from real operational sites. Polygon-level annotations were produced by professional annotators working directly from original CAD design drawings of each facility, yielding geometrically accurate element boundaries that reflect the as-built state of the sites. Each layout is composed of elements drawn from 14 semantic classes organized into four functional groups: (i) site premise and access: referring to layout premise $\mathcal{P}$, ENTRANCE and EXIT; (ii) service infrastructure: three service unit variants differentiated by vehicle size (CSA, BSA, HGVS), an electric vehicle charging area (EVSA), an underground service facility (USF), and an underground access point (UAP); (iii) building and auxiliary services: a retail building (SHOP); and (iv) parking: four bay variants: car (CP), bicycle (BP), priority (PP), and heavy-goods vehicle (HGVP). Element composition is instance-specific: not all 14 classes appear in every layout, introducing compositional variability and class imbalance characteristic of real facility layout design. An example layout is illustrated in Fig. 1 (left). The layouts are governed by a shared design manual $\mathcal{M}$ (as discussed in Section 3, with details in Appendix D.4), released alongside the dataset. The full routing scenarios and soft constraint specification are detailed in Appendices D.5 and D.4.2. In contrast to existing web and document layout benchmarks, **ALPS-Bench** offers challenging design through variable premise shape, varying numbers of elements and access points, and an imbalanced class distribution.

Data Split and Structure. The 1,000 layouts are partitioned into a development set of 900 instances ($\mathcal{D}^{\text{dev}}$) and a held-out test set of 100 instances ($\mathcal{D}^{\text{test}}$). Development instances are released with full annotations (input requirements $\mathcal{I}$ together with the ground-truth configuration $\mathbf{q}^\dagger$) making this set suitable for training, in-context learning, and hyperparameter selection. For test instances, only the input requirements $\mathcal{I}$ are made public; ground-truth configurations are withheld to support standardized automated evaluation on a hosted online server. Further details on dataset statistics and annotation format can be found in Appendix D.3 and D.2, respectively.

4.1 Evaluation and Benchmarking

To evaluate newly designed layouts, we propose an instance-specific scorer instead of metrics such as IoU or FID, which can penalize design innovation by promoting prior matching. The scoring function evaluates generated instances from two perspectives: (i) satisfaction of design manual ($\mathcal{M} = \{\mathcal{HC}, \mathcal{SC}\}$); and (ii) comparison against withheld ground-truth layouts $\mathbf{q}^\dagger$ in terms of routing efficiency. Let us delve deeper into the evaluation metrics.

Hard Constraints Metrics (HCM). To evaluate satisfaction of the hard constraints ($\mathcal{HC}$) of the design manual ($\mathcal{M}$) in Section 3, this metric investigates the existence and correct sizing of all elements specified by input requirement $\mathcal{I}$; containment ($\Omega_i \subseteq \mathcal{P}, \forall i$); non-overlap ($\Omega_i \cap \Omega_j = \emptyset$, $\forall i \neq j$); and routing feasibility across scenarios in $\mathcal{R}$. While the other conditions are straightforward to measure, routing feasibility is a non-trivial assessment task. To assess unobstructed navigation for a routing scenario, we adopt a simple Voronoi-based motion-planning approach [28, 6]. We first inflate the borders of obstacles (i.e., $e_i \notin \mathcal{R}$) by a scenario-dependent safety margin ϵ. We then sample seed points on the inflated borders and use them to construct a Voronoi diagram. The Voronoi medial axes are used to identify paths with maximum clearance. This is shown in Fig. 1 (right), with further details in Appendix D.6. Missing or incorrectly sized elements result in HCM $= 0$. Otherwise, the score is a continuous value in $(0, 1]$, computed as $\text{HCM} = \alpha_{\text{g}} \times \text{HC}_{\text{g}} + \alpha_{\text{f}} \times \text{HC}_{\text{f}}$, where the

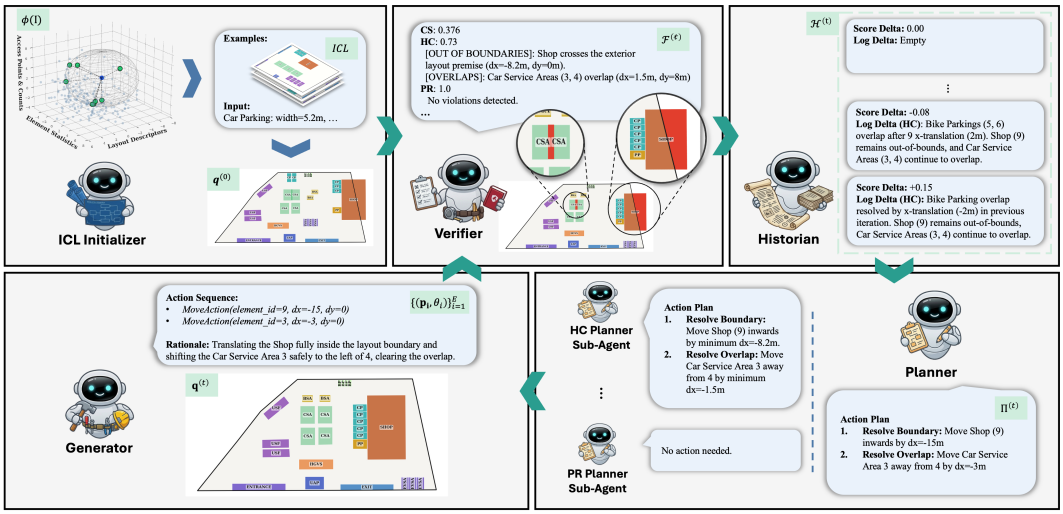

Figure 2: The interplay and cooperation among the agents in CANDO. Upon an ICL Initialization, Verifier, Historian, Planner and Generator communicate in a structured manner within a loop.

geometric validity score HC_g proportionally penalizes out-of-boundary and overlapping areas, and the flow score HC_f is the fraction of feasible routing scenarios. We use $\alpha_\text{g} = 0.2$ and $\alpha_\text{f} = 0.8$, assigning more weight to valid routing because it is considerably more complex to address.

Soft Constraint Metrics (SCM). To evaluate satisfaction of the soft constraints ($\mathcal{SC}$) part of the design manual ($\mathcal{M}$), this metric investigates three aspects. Proximity (PR) $\in [0, 1]$ checks stacking/adjacency, minimum clearance, and maximum inter-element distance requirements, penalizing each violation at -0.1. Compositionality (CO) $\in [0, 1]$ checks whether elements of each semantic class form the spatial formations required by the design manual (e.g., line arrays, matrix grids), also penalizing fragmentation and wrong formations at -0.1 per violation. Orientation (OR) $\in [0, 1]$ checks intra-group angular alignment, with penalties scaling continuously with angular deviation. The score is a continuous value in $[0, 1]$, computed as $\text{SCM} = \alpha_\text{p} \times \text{PR} + \alpha_\text{c} \times \text{CO} + \alpha_\text{o} \times \text{OR}$. We currently use $\alpha_\text{p} = 0.6$, $\alpha_\text{c} = 0.3$ and $\alpha_\text{o} = 0.1$, reflecting the proportional number of soft constraint rules for each category included in $\mathcal{M}$.

Ground Truth Metrics (GTM). To go beyond constraint satisfaction, this metric compares generated layouts against the withheld ground-truth configurations $\mathbf{q}^\dagger$ along three operational dimensions: path length (PL), maneuverability (MN), and path clearance (PC), each normalized to $[0, 1]$ with 0.5 indicating equivalence to the ground truth. PL is scored as $\overline{\ell}_\text{gt}/(\overline{\ell}_\text{gt} + \overline{\ell}_\text{gen})$; MN as $\overline{n}_\text{gt}/(\overline{n}_\text{gt} + \overline{n}_\text{gen})$, where $\overline{n}$ is the average number of turns; and PC as $\overline{c}_\text{gen}/(\overline{c}_\text{gt} + \overline{c}_\text{gen})$, where $\overline{c}$ is the average minimum distance to obstacles along the paths. Subscripts $(.)_\text{gt}$ and $(.)_\text{gen}$ refer to ground-truth and generated layouts, respectively. Values above 0.5 indicate shorter routes, fewer turns, or wider clearances relative to the ground truth. Invalid scenarios receive $\text{PL} = \text{MN} = \text{PC} = 0$.

Aggregate Score (AGG). With all sub-metrics bounded in $[0, 1]$, we define the final scores:

$$\text{AGG} = \tfrac{1}{2}(\text{CS} + \text{RE}),\ \text{CS} = \begin{cases} \text{SCM} & \text{if HCM} = 1 \\ \alpha \times \text{HCM} \times \text{SCM} & \text{if HCM} < 1 \end{cases},\ \text{RE} = \tfrac{1}{3}(\text{PL} + \text{MN} + \text{PC}). \quad (3)$$

Our constraint satisfaction metrics are summarized in CS, where invalid final layouts ($\text{HCM} < 1$) are penalized with a capped score of α. We set $\alpha = 0.5$ to emphasize that invalid layouts are undesirable. The routing efficiency metrics are summarized in RE, and CS and RE are averaged to produce the aggregate score (AGG). All scores are computed per layout and then averaged over the relevant split.

5 Methodology

To solve the optimization problem introduced in Section 3 and instantiated with **ALPS-Bench**, we propose Cooperative Agentic Network for Layout Design Optimization (CANDO), a multi-agent frame-

work for constraint-aware layout synthesis. As motivated in Section 1, this cooperative architecture distributes reasoning tasks across specialized agents to avoid the failure modes of single-model generation and to enable processing with resource-constrained agents (with smaller backbones). This design choice is supported by both prior art [30, 15, 53, 50] and our experiments in Section 6. The proposed solution operates in two phases. First, an *in-context learning initializer (ICL Initializer)* produces a seed configuration $\mathbf{q}^{(0)}$ by conditioning an LLM on structurally similar layouts retrieved from the development data. Second, four specialized agents (*Verifier*, *Historian*, *Planner*, and *Generator*) cooperate to refine $\mathbf{q}^{(0)}$ toward $\mathbf{q}^{\star}$ through a parallel-branching iterative loop adapted from CoDe's blockwise sampling scheme for diffusion models [46]. Crucially, since language models struggle with spatial understanding and coordinate arithmetic [56], `CANDO` deliberately offloads low-level geometric computations to deterministic tools.

The two-phase design reflects the asymmetric interplay between feasibility and design quality, as hard-constraint violations would render soft-constraint quality irrelevant. The proposed solution design and inner-mechanics of the agentic cooperation are illustrated in greater detail in Fig. 2. The overall optimization process is summarized in Algorithm 1.

In-Context Learning Initialization. Given an input requirement $\mathcal{I}_{\text{new}}$ (Definition 1), the ICL Initializer agent produces an initial configuration $\mathbf{q}^{(0)}$ by conditioning layout synthesis on structurally similar examples retrieved from $\mathcal{D}^{\text{dev}}$. The example retrieval process starts by embedding new input requirements $\mathcal{I}_{\text{new}}$ into a feature vector $\phi(\mathcal{I}_{\text{new}}) \in \mathbb{R}^d$ covering four aspects: (i) per-class element counts and area coverages, (ii) access-point counts and their positions, (iii) element size statistics, and (iv) layout descriptors (area, perimeter, compactness). The most similar layouts to the query are retrieved with a weighted kNN search over the embedding space representing $\{\phi(\mathcal{I}) \mid \mathcal{I} \in \mathcal{D}^{\text{dev}}\}$. The retrieved examples are serialized as input–output pairs (input: $\mathcal{I}$, output: $\mathbf{q}$) and provided as context to the agent, which is conditioned on $\mathcal{I}_{\text{new}}$. Next, K_{init} candidate layouts are generated in parallel and ranked using a geometric hard-constraint satisfaction proxy:

$$\sigma(\mathbf{q}) \coloneqq \frac{1}{1 + V_{\text{out}}(\mathbf{q}) + V_{\text{ov}}(\mathbf{q})},\ V_{\text{out}}(\mathbf{q}) \coloneqq \frac{1}{|\mathcal{P}|}\sum_{i=1}^{E} |\Omega_i \setminus \mathcal{P}|,\ V_{\text{ov}}(\mathbf{q}) \coloneqq \frac{1}{|\mathcal{P}|}\sum_{i<j\leq E} |\Omega_i \cap \Omega_j| \quad (4)$$

where $V_{\text{out}}(\mathbf{q})$ and $V_{\text{ov}}(\mathbf{q})$ are dimensionless out-of-premise and unique-pair overlap violations, normalized by premise area. Thus, $\sigma(\mathbf{q}) = 1$ exactly when the candidate satisfies containment and pairwise non-overlap. We rank the candidates by $\sigma(\cdot)$ and select the highest-scoring one. The result is returned as $\mathbf{q}^{(0)}$. Further details on specific feature definitions and the weighted kNN approach can be found in Appendix F.1.

Verification. It lies at the core of our proposed approach, providing *reward signals* that guide the refinement of intermediate layouts. Given K candidates in $\mathcal{Q}^{(t)}$, the Verifier evaluates them against the design manual $\mathcal{M}$ using dedicated deterministic tools and selects the best-scoring candidate $\bar{\mathbf{q}}^{(t)}$. The Verifier and benchmark evaluator intentionally share ground-truth-free design-manual checks (HC_g, HC_f, CO, PR, OR), but final scores are recomputed independently and the optimization never observes a held-out target layout. In particular, PL, MN, and PC compare against $\mathbf{q}^{\dagger}$ and are available only to the benchmark evaluator; hence, RE and AGG are not optimization signals. For richer diagnostics, the Verifier treats routing separately rather than integrating it into HCM, reports the five constraint aspects and CS, and aggregates fine-grained diagnostics (e.g., obstacles blocking a route) into $\mathcal{F}^{(t)}$ for the Historian and Planner.

History Management. The Historian agent maintains a running record of how the layout design evolves. After each step t, it computes the change in feedback and annotates it with a concise description of the spatial changes that led to it. A sliding window containing at most W recent changes, $\mathcal{H}^{(t)} = \{\Delta\mathcal{F}^{(s)}\}_{s=\max(1,t-W+1)}^{t}$, is passed to the Planner, grounding its decisions in the evolution of feedback over time rather than only its current state.

Planner. The Planner agent drafts its remediation plan along the same five evaluation categories used by the Verifier. Category-level plans are produced by the same agent, if the agent is capable of handling the full context (with a large backbone), or decomposed and processed in parallel by specialized planning sub-agents/workers, with task-specific tools, and then reconciled by a supervisor

into a remediation plan $\Pi^{(t)}$ at each time step t (as shown in Fig. 2). To resolve cross-category conflicts and prioritize repairs, the supervisor grounds its decisions in a *multi-modal* input context defined by four complementary pieces of information: (i) the targeted diagnostic feedback from the specialist workers, (ii) the history $\mathcal{H}^{(t)}$, (iii) the geometric coordinate representation of the layout, and (iv) its visual rendering created by a simple plotting tool. By incorporating a subset or all of these signals, the supervisor formulates a coherent global intervention plan.

Generator. This agent translates the remediation plan $\Pi^{(t)}$ into a set of K revised candidate configurations $\mathcal{Q}^{(t)} = \{\mathbf{q}_k^{(t)}\}_{k=1}^{K}$ by applying a sequence of modifications to $\bar{\mathbf{q}}^{(t-1)} = \{(\mathbf{p}_i, \theta_i)\}_{i=1}^{E}$ (Eq. (1)). The Generator adopts a strategy-diversified ensemble generation, with candidates either applying the plan conservatively through local repairs, prioritizing a single category or allowing larger layout-wide modifications. This ensemble design is important in practice because different layouts benefit from different remediation strategies. The strategy ensemble and their detailed definitions are discussed in detail in Appendix F.2.

Cooperative Refinement Loop. Starting from $\bar{\mathbf{q}}^{(0)}$, CANDO refines the layout through a parallel-branching loop that adapts CoDe's blockwise sampling scheme from diffusion models [46], as summarized in Algorithm 1. Until the budget of T refinement steps is exhausted, in every iteration, CANDO spawns N independent exploration branches from $\bar{\mathbf{q}}^{(t)}$ (Line 5). Every branch has its own local history ($\mathcal{H}_n^{(t)}$) and pursues its own trajectory of B sequential steps. Each step corresponds to an execution of the history-aware planning-generation-verification sequence (Lines 6–9). At the end of each block (B steps) a BoN reduce operation (Line 12) selects the branch with the highest score (the new $\bar{\mathbf{q}}^{(t)}$), as well its corresponding history and feedback (the new $\mathcal{H}^{(t)}$ and $\mathcal{F}^{(t)}$, respectively), and passes it onto the next step where it will be replicated N times to start the next block (Line 4). The (block-wise) BoN approach enables the system to simultaneously explore structurally different refinements without committing prematurely to any single branch. Note that setting $B = T$ transforms CoDe back to BoN. The interested reader is referred to Appendix F for more elaborate discussions on the individual agents.

Algorithm 1 CANDO

Require: $\mathcal{I}, \mathcal{D}^{\text{dev}}, N, B, K, W, T$
1: $\mathcal{H}^{(0)} \leftarrow \emptyset;\ t \leftarrow 0$
2: $(\bar{\mathbf{q}}^{(0)}, \mathcal{F}^{(0)}) \leftarrow \textsc{Verifier}(\{\textsc{ICLInit}(\mathcal{I}, \mathcal{D}^{\text{dev}})\})$
3: **while** $t \leq T - B$ **do**
4: $\quad\{\mathcal{H}_n^{(t)}, \mathcal{F}_n^{(t)}, \mathbf{q}_n^{(t)}\}_{n=1}^{N} \leftarrow \textsc{Replicate}(\mathcal{H}^{(t)}, \mathcal{F}^{(t)}, \bar{\mathbf{q}}^{(t)})$
5: $\quad$**for** $b = 0, \ldots, B-1$ **do** $\quad\triangleright\ \forall n \in [N]$, in parallel
6: $\quad\quad\mathcal{H}_n^{(t+b+1)} \leftarrow \textsc{Historian}(\mathcal{H}_n^{(t+b)}, \mathcal{F}_n^{(t+b)}, W)$
7: $\quad\quad\Pi_n^{(t+b+1)} \leftarrow \textsc{Planner}(\mathbf{q}_n^{(t+b)}, \mathcal{F}_n^{(t+b)}, \mathcal{H}_n^{(t+b+1)})$
8: $\quad\quad\mathcal{Q}_n^{(t+b+1)} \leftarrow \textsc{Generator}(\mathbf{q}_n^{(t+b)}, \Pi_n^{(t+b+1)}, K)$
9: $\quad\quad(\bar{\mathbf{q}}_n^{(t+b+1)}, \mathcal{F}_n^{(t+b+1)}) \leftarrow \textsc{Verifier}(\mathcal{Q}_n^{(t+b+1)})$
10: $\quad$**end for**
11: $\quad t \leftarrow t + B$
12: $\quad(\bar{\mathbf{q}}^{(t)}, \mathcal{F}^{(t)}, \mathcal{H}^{(t)}) \leftarrow \textsc{BoN}\left(\{\bar{\mathbf{q}}_n^{(t)}, \mathcal{F}_n^{(t)}, \mathcal{H}_n^{(t)}\}_{n=1}^{N}\right)$
13: **end while**, **return** $\bar{\mathbf{q}}^{(T)}$

6 Experiments

Experimental Setup. We assess CANDO on our proposed **ALPS-Bench** and three public benchmarks: PubLayNet [59], RICO [7], and PKU-PosterLayout [17]. For ALPS-Bench, the problem setting and corresponding evaluation metrics are defined in Sections 3 and 4, respectively. For PubLayNet and RICO, we evaluate all tasks defined and adopted by prior art: type-conditioned generation (Gen-T), type/size-conditioned generation (Gen-TS), relational generation (Gen-R), refinement, and completion (latter three in Appendix E.4). A simplified version of CANDO is applied on the public benchmarks because not all agents are needed; its complete configuration is given in Appendix E.2. For PubLayNet and RICO, we follow the evaluation criteria in the literature with the adjusted mIoU (as per [25]), alignment ("Align."), overlap, and FID scores. For the latter three, we adopt the implementations of [22]. For most of our experiments we opt for Qwen3.6 [40] (Qwen/Qwen3.6-35B-A3B-FP8) as a backbone for our agents. This variant of Qwen is a lightweight, yet powerful open-source Mixture-of-Experts (MoE) [21, 9] model of 35B total parameters and 3B active parameters. For quantitative evaluations, we used NVIDIA H100 GPUs with 80GB of VRAM. For our experiments with **ALPS-Bench**, we also provide experimental results with GPT-5.2 (API access via a cloud provider), a model with strong reasoning capabilities and orders of magnitude larger size.

Results on ALPS-Bench. Table 1 summarizes the performance of CANDO using our adaptation of CoDe's blockwise sampling procedure [46] with two backbone variants (Qwen and GPT-5.2).

Table 1: Quantitative results of layout generation models evaluated on the **ALPS-Bench** test set ($\mathcal{D}^{\text{test}}$). † denotes methods trained or fine-tuned on ALPS-Bench; all other methods are training-free with respect to ALPS-Bench. Symbols indicate the generation paradigm: ♦ diffusion-based, ▲ single-LLM/MLLM, and • agentic.

Method	Summary Statistics			T	Avg. Iterations to Comply	Time Complexity	Compliance (%)
	AGG↑	CS↑	RE↑				
▲ Qwen-LoRA-ALPS†	$0.07_{\pm 0.12}$	$0.01_{\pm 0.08}$	$0.13_{\pm 0.22}$	1	–	τ	–
♦ LayoutDM† [19]	$0.13_{\pm 0.13}$	$0.01_{\pm 0.03}$	$0.24_{\pm 0.25}$	–	–	–	–
▲ LayoutPrompter (GPT-5.2) [32]	$0.17_{\pm 0.11}$	$0.01_{\pm 0.06}$	$0.33_{\pm 0.21}$	1	–	τ	–
▲ LayoutCoT (GPT-5.2) [44]	$0.19_{\pm 0.20}$	$0.13_{\pm 0.24}$	$0.25_{\pm 0.24}$	4	–	4τ	–
• Single-Agent (GPT-5.2)	$0.40_{\pm 0.27}$	$0.47_{\pm 0.37}$	$0.32_{\pm 0.22}$	200	80.6	80.6τ	22.4
• CANDO (BoN/GPT-5.2)	$0.59_{\pm 0.25}$	$0.77_{\pm 0.36}$	$0.41_{\pm 0.17}$	20	8.55	$8.55 \times 4\tau = 34.2\tau$	60.6
• CANDO (CoDe/GPT-5.2)	$0.60_{\pm 0.25}$	$0.78_{\pm 0.35}$	$\mathbf{0.42_{\pm 0.17}}$	20	8.81	$8.81 \times 4\tau = 35.24\tau$	63.6
• CANDO (CoDe/Qwen3.6)	$\mathbf{0.62_{\pm 0.19}}$	$\mathbf{0.83_{\pm 0.25}}$	$0.41_{\pm 0.16}$	20	**5.09**	$5.09 \times 4\tau = \mathbf{20.36}\tau$	**74.0**

ALPS-Bench goes beyond standard generation tasks conditioned on type and size through an elaborate design manual. Therefore, few baselines apply directly to the newly defined tasks without considerable architectural modification and retraining. To compare against training-based approaches, we additionally adapt and train LayoutDM [19] from scratch on **ALPS-Bench** and fine-tune Qwen3-VL-8B-Instruct [57] using LoRA-SFT [18]. LayoutDM requires an adaptation of its native axis-aligned layout representation to accommodate the polygonal geometry and element orientations of **ALPS-Bench**, whereas Qwen-LoRA-ALPS generates the complete polygonal layout in a single pass conditioned on the rendered site and structured requirements. Adaptation details for both baselines are provided in Appendix E.1. We further compare against the pioneering LLM-based approaches LayoutPrompter [32] and LayoutCoT [44], as well as our implementation of a Single-Agent using a strong backbone (GPT-5.2) and equipped with the evaluation scores and toolset devised for CANDO.

The summary statistics demonstrate that CANDO variants substantially outperform all considered baselines, including the two models trained specifically on **ALPS-Bench**, while requiring an order of magnitude smaller total number of refinement steps (T) than the Single-Agent baseline, with a time-evolution graph shown in Fig. 3. LayoutDM frequently violates geometric hard constraints because its bounding-box-based representation only approximates the true polygonal site geometry, while the single-pass Qwen-LoRA-ALPS can capture high-level compositional patterns but struggles with exact containment, non-overlap, and routing feasibility. These results suggest that dataset-specific training alone is insufficient for the hard geometric constraints of ALPS-Bench and further highlight the importance of iterative, tool-grounded verification. Additionally, paired, instance-level two-sided Wilcoxon tests with Holm correction and paired-bootstrap 95% CIs over the 100 test instances confirm that both CANDO variants significantly outperform all considered baselines on AGG, CS, and RE (all $p_{\text{Holm}} \leq 0.0059$), while the Qwen and GPT-5.2 variants are statistically comparable across all three metrics (p_{Holm}=1.000). Full results are reported in Appendix E.8.

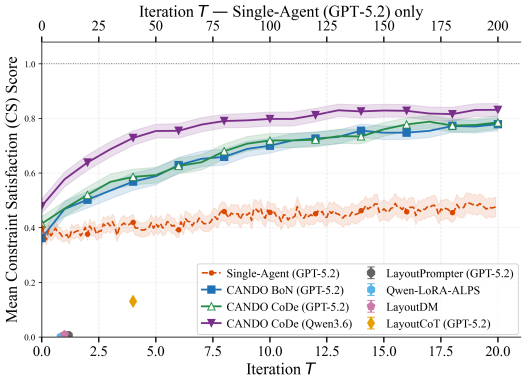

Figure 3: Mean CS Score vs. T for Table 1

For the sake of fair comparison, we also provide time complexity of each approach averaged over the samples in $\mathcal{D}^{\text{test}}$. To measure time complexity, we assume a single LLM inference call to incur a fixed complexity of τ. To provide a more complete picture, we compute the average number of iterations per method to reach compliance (on the samples achieving that). Considering a factor of $\times 4$ taking into account the number of sequential calls to each agent in CANDO, we arrive at the summary provided in the table. The results show that CANDO requires the same order of complexity (half or even less, to be precise). This underscores that despite the increased operational complexity per step, the parallelized nature of CANDO yields a competitive overall complexity that scales more favorably than the extensive sequential Single-Agent counterpart.

Results on PubLayNet and RICO. To further assess the efficacy of CANDO beyond the proposed **ALPS-Bench**, we extended our evaluations against state-of-the-art layout generation models on two widely adopted benchmarks: PubLayNet and RICO. Table 2 summarizes the performance for type-

Table 2: Quantitative results on PubLayNet and RICO datasets. Extended version in Appendix E.4.

Task	Method	RICO [7]				PubLayNet [59]			
		mIoU↑	FID↓	Align↓	Over↓	mIoU↑	FID↓	Align↓	Over↓
Gen-T	LayoutGAN++ [25]	0.298	5.954	0.261	0.620	0.297	14.875	0.124	0.148
	BLT [26]	0.216	25.633	0.150	0.983	0.140	38.684	0.036	0.196
	LayoutFormer++ [22]	0.432	**1.096**	0.230	0.530	0.348	8.411	0.020	0.008
	LayoutPrompter [32]	0.604	2.134	0.002	0.029	0.619	1.648	0.001	0.004
	LayoutCoT [44]	0.719	2.189	0.001	0.022	0.772	**1.227**	**0.000**	0.002
	CANDO (ours)	**0.866**	2.101	**0.000**	**0.005**	**0.893**	3.455	**0.000**	**0.000**
Gen-TS	BLT [26]	0.604	0.951	0.181	0.660	0.428	7.914	0.021	0.419
	LayoutFormer++ [22]	0.620	**0.757**	0.202	0.542	0.471	**0.720**	0.024	0.037
	LayoutPrompter [32]	0.707	3.072	0.002	0.051	0.727	1.863	0.001	0.076
	LayoutCoT [44]	0.795	3.506	**0.000**	**0.031**	0.847	1.430	**0.000**	0.004
	CANDO (ours)	**0.879**	1.371	**0.000**	0.052	**0.881**	2.023	**0.000**	**0.000**

conditioned (Gen-T) and type/size-conditioned (Gen-TS) layout generation tasks. CANDO achieves the highest mIoU on both settings by a substantial margin, and perfect alignment scores across both benchmarks and tasks. This is noteworthy given that here CANDO runs on a comparatively lightweight open-source backbone Qwen with 35B parameters compared to GPT-4 used in LayoutCoT [44] and the most recent (and the best) reported results for LayoutPrompter [32]. Across all five RICO tasks, CANDO also improves overlap over LayoutCoT in four cases, with Gen-TS as the exception (Appendix E.4). These alignment and overlap results reflect CANDO's verification and tool-calling strategies. By offloading axis-alignment and overlap checks to dedicated geometric tools, rather than relying on the (M)LLM to reason about coordinates, CANDO handles such violations precisely.

The FID scores here present a more nuanced picture. Despite leading on mIoU, CANDO does not consistently achieve competitive FID, ranking second or last depending on the setting. This inconsistency is not incidental but rather symptomatic of the broader limitation of distributional evaluation we discussed in Sections 1 and 4. We argue that a method that produces geometrically precise layouts that satisfy desired constraints and requirements yet happens to deviate from the distribution of the training data will incur a high FID penalty, even if its outputs are arguably of higher design quality. Methods such as LayoutFormer++, which are trained explicitly against the prior distribution, are inherently better at minimizing FID, even though as the mIoU results suggest, at the cost of spatial accuracy. Notably, this is known in the context of reward-aligned foundation models (analogous to meeting design requirements through a verification scheme in our setting) where more innovative samples can be generated at the cost of larger KL divergence from prior data [20, 46].

Results on PKU-PosterLayout. To further assess the generalizability of CANDO, we additionally adapt and evaluate it on PKU-PosterLayout, considering its controllable generation tasks. Table 3 reports results for type-conditioned (Gen-T) and type/size-conditioned (Gen-TS) layout generation tasks. Results for relationship-conditioned generation, completion, and refinement are deferred to Appendix E.5. Across both tasks, CANDO exhibits particularly strong performance on metrics directly associated with explicit spatial constraint satisfaction. Specifically, it attains the best Strict Underlay scores (1.000 and 0.990) while simultaneously reducing overlap to 0.0009 and 0.0, respectively, outperforming both RALF [16] and UniLayDiff [33] on these metrics. Conversely, CANDO obtains higher FID scores and does not consistently improve occlusion or readability. This behavior is consistent with the observations on PubLayNet and RICO: CANDO's verification-driven generation procedure explicitly optimizes compliance with geometric and relational requirements rather than reproducing the empirical layout distribution of the dataset, as measured by distribution-sensitive metrics such as FID.

Table 3: Quantitative results on PKU-PosterLayout dataset. Additional tasks are reported in Appendix E.5.

Task	Method	Occ↓	Rea↓	Und_S↑	Ove↓	FID↓
Gen-T	RALF [16]	0.136	**0.0127**	0.890	0.0091	**2.36**
	UniLayDiff [33]	**0.122**	0.0129	0.963	0.0018	3.09
	CANDO (ours)	0.144	0.0196	**1.000**	**0.0009**	9.83
Gen-TS	RALF [16]	0.138	**0.0123**	0.876	0.0113	**0.61**
	UniLayDiff [33]	**0.128**	0.0132	0.918	0.0046	0.66
	CANDO (ours)	0.145	0.0197	**0.990**	**0.0000**	4.47

Component Ablation. Table 4 assesses the contribution of the main architectural components of CANDO. We define the *regression rate* as the percentage of refinement iterations in which the verification score decreases after applying the proposed changes, capturing how often a refinement step inadvertently introduces or worsens constraint violations. CoDe and tool-grounded verification have the most pronounced impact. Replacing CoDe with a single refinement trajectory reduces AGG from

Table 4: Component ablation of CANDO. Regression rate denotes the percentage of refinement iterations for which the verification score decreases after applying the proposed changes.

Variant	AGG↑	CS↑	RE↑	Verifier-Compliant Layouts	Mean Iterations	Regression Rate
CANDO (full)	$0.62_{\pm 0.19}$	$0.83_{\pm 0.25}$	$0.41_{\pm 0.16}$	74%	13.28	7.0%
w/o CoDe	$0.35_{\pm 0.30}$	$0.38_{\pm 0.45}$	$0.32_{\pm 0.23}$	44%	16.75	6.3%
w/o Historian	$0.61_{\pm 0.23}$	$0.80_{\pm 0.34}$	$0.42_{\pm 0.16}$	59%	13.35	6.7%
w/o ICL	$0.57_{\pm 0.26}$	$0.75_{\pm 0.37}$	$0.40_{\pm 0.19}$	53%	14.39	5.8%
w/o Planner	$0.62_{\pm 0.22}$	$0.82_{\pm 0.32}$	$0.43_{\pm 0.15}$	51%	14.63	6.7%
w/o Strategies	$0.61_{\pm 0.23}$	$0.80_{\pm 0.32}$	$0.42_{\pm 0.16}$	53%	14.60	18.4%
w/o Tool-grounded Verifier	$0.27_{\pm 0.23}$	$0.28_{\pm 0.27}$	$0.25_{\pm 0.24}$	95%*	4.63	19.6%
w/o Visual Rendering	$0.61_{\pm 0.24}$	$0.80_{\pm 0.35}$	$0.42_{\pm 0.16}$	61%	13.35	7.3%

*Without tool access, the Verifier acts solely as an LLM-as-a-Judge, inflating its scores.

0.62 to 0.35, CS from 0.83 to 0.38, and verifier-compliant layouts from 74% to 44%, while increasing the mean number of iterations from 13.28 to 16.75. This demonstrates the importance of parallel exploration for escaping difficult constraint violations rather than committing to a single refinement trajectory. Tool-grounded verification is similarly critical. Without deterministic verification tools, the LLM-based Verifier declares 95% of layouts compliant and terminates after only 4.63 iterations on average, despite the independent evaluation yielding only 0.27 AGG and 0.28 CS. The corresponding regression rate also increases from 7.0% to 19.6%. This discrepancy highlights that an LLM-as-a-Judge can substantially overestimate constraint satisfaction and corroborates the importance of grounding feedback in deterministic tools.

Strategy diversification primarily affects the stability of refinement. Removing the alternative refinement strategies leaves the average scores comparatively unchanged, but increases the regression rate from 7.0% to 18.4% and reduces compliance from 74% to 53%. Thus, a single generic repair strategy is substantially more likely to resolve one violation while introducing another. The distribution of winning strategies, reported in Appendix E.6, further shows that no individual strategy dominates the refinement process. ICL provides the next most pronounced benefit, with its removal reducing AGG from 0.62 to 0.57, CS from 0.83 to 0.75, and compliance from 74% to 53%, while requiring more refinement iterations. The remaining components primarily improve robustness rather than mean performance. Notably, removing the Planner leaves mean AGG nearly unchanged (0.62) but reduces compliance from 74% to 51%, suggesting that direct Verifier feedback can often repair individual violations but is less reliable at jointly reconciling all constraints.

Hyperparameter Tuning. Detailed hyperparameter ablations for **ALPS-Bench** are reported in Appendix E.7. Based on a development-set sweep using Qwen for cost efficiency, we select $N{=}4$, $B{=}10$, $K{=}6$, and $W{=}6$, which achieve 0.76 CS and 0.59 AGG. We retain this configuration across backbone variants and increase the refinement budget to $T{=}20$ for the reported ALPS-Bench experiments. Appendix E.2 details the fixed configurations used on all public benchmarks.

7 Concluding Remarks

We introduced **ALPS-Bench**, a benchmark of 1,000 professionally annotated facility layouts with an instance-specific scoring protocol that measures actual constraint satisfaction, and CANDO, a training-free cooperative agentic framework that refines layouts through a verification-grounded loop. CANDO establishes a strong baseline on **ALPS-Bench** and achieves state-of-the-art mIoU and alignment on PubLayNet and RICO with a comparatively lightweight backbone. On PKU-PosterLayout, it obtains the lowest overlap across all five tasks and the highest strict-underlay score in four of five tasks. These results corroborate that cooperative agentic design is a broadly effective recipe for constraint-aware layout synthesis. We hope **ALPS-Bench** shifts the focus of layout generation research beyond the current regime and toward the operational complexity of real-world design, and that CANDO motivates further work in this direction, positioning cooperative agentic design with tool-grounded verification as a strong recipe for constraint-aware synthesis.

Limitations. The proposed **ALPS-Bench** is built from outdoor service facility layouts, and transferring CANDO to a new domain requires a corresponding design manual and, if ICL is used, a development corpus. The parallel-branching strategy also incurs more (M)LLM calls than single-agent baselines, with a clear performance-cost tradeoff (Table 15). Finally, CANDO does not optimize for distributional similarity, yielding strong mIoU but weaker FID on PubLayNet, RICO, and PKU.

Acknowledgments and Disclosure of Funding

Acknowledgments. The authors thank Anuj Singh (formerly Shell, now TU Delft) for the initial insightful technical ideations and help with data collection, Stylianos Poulakakis-Daktylidis for his assistance with data curation, as well as Prof. Hamed Pirsiavash (UC Davis) for early insightful discussions.

Funding in direct support of this work. This work was funded by Shell.

Competing interests. The authors declare no competing interests.

References

[1] Mohamed Abouagour and Eleftherios Garyfallidis. Resplan: A large-scale vector-graph dataset of 17,000 residential floor plans. *arXiv preprint arXiv:2508.14006*, 2025.

[2] Ahmad Beirami, Alekh Agarwal, Jonathan Berant, Alexander D'Amour, Jacob Eisenstein, Chirag Nagpal, and Ananda Theertha Suresh. Theoretical guarantees on the best-of-n alignment policy. *arXiv preprint arXiv:2401.01879*, 2024.

[3] Boyuan Chen, Zhuo Xu, Sean Kirmani, Brain Ichter, Dorsa Sadigh, Leonidas Guibas, and Fei Xia. Spatialvlm: Endowing vision-language models with spatial reasoning capabilities. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 14455–14465, 2024.

[4] Weixing Chen, Dafeng Chi, Yang Liu, Yuxi Yang, Yexin Zhang, Yuzheng Zhuang, Xingyue Quan, Jianye Hao, Guanbin Li, and Liang Lin. Autolayout: Closed-loop layout synthesis via slow-fast collaborative reasoning. *arXiv preprint arXiv:2507.04293*, 2025.

[5] Weize Chen, Yusheng Su, Jingwei Zuo, Cheng Yang, Chenfei Yuan, Chi-Min Chan, Heyang Yu, Yaxi Lu, Yi-Hsin Hung, Chen Qian, et al. Agentverse: Facilitating multi-agent collaboration and exploring emergent behaviors. In *The Twelfth International Conference on Learning Representations*, 2023.

[6] Howie Choset and Joel Burdick. Sensor based planning. i. the generalized voronoi graph. In *Proceedings of 1995 IEEE international conference on robotics and automation*, volume 2, pages 1649–1655. IEEE, 1995.

[7] Biplab Deka, Zifeng Huang, Chad Franzen, Joshua Hibschman, Daniel Afergan, Yang Li, Jeffrey Nichols, and Ranjitha Kumar. Rico: A mobile app dataset for building data-driven design applications. In *Proceedings of the 30th annual ACM symposium on user interface software and technology*, pages 845–854, 2017.

[8] Yilun Du, Shuang Li, Antonio Torralba, Joshua B Tenenbaum, and Igor Mordatch. Improving factuality and reasoning in language models through multiagent debate. In *Forty-first international conference on machine learning*, 2024.

[9] William Fedus, Barret Zoph, and Noam Shazeer. Switch transformers: Scaling to trillion parameter models with simple and efficient sparsity. *Journal of Machine Learning Research*, 23 (120):1–39, 2022.

[10] Weixi Feng, Wanrong Zhu, Tsu-jui Fu, Varun Jampani, Arjun Akula, Xuehai He, Sugato Basu, Xin Eric Wang, and William Yang Wang. Layoutgpt: Compositional visual planning and generation with large language models. *Advances in Neural Information Processing Systems*, 36:18225–18250, 2023.

[11] Jialin Gao, Donghao Zhou, Mingjian Liang, Lihao Liu, Chi-Wing Fu, Xiaowei Hu, and Pheng-Ann Heng. Disco-layout: Disentangling and coordinating semantic and physical refinement in a multi-agent framework for 3d indoor layout synthesis. *arXiv preprint arXiv:2510.02178*, 2025.

[12] John C Gower. Generalized procrustes analysis. *Psychometrika*, 40(1):33–51, 1975.

[13] Julian Jorge Andrade Guerreiro, Naoto Inoue, Kento Masui, Mayu Otani, and Hideki Nakayama. Layoutflow: flow matching for layout generation. In *European Conference on Computer Vision*, pages 56–72. Springer, 2024.

[14] Kamal Gupta, Justin Lazarow, Alessandro Achille, Larry S Davis, Vijay Mahadevan, and Abhinav Shrivastava. Layouttransformer: Layout generation and completion with self-attention. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 1004–1014, 2021.

[15] Sirui Hong, Mingchen Zhuge, Jonathan Chen, Xiawu Zheng, Yuheng Cheng, Jinlin Wang, Ceyao Zhang, Zili Wang, Steven Ka Shing Yau, Zijuan Lin, et al. Metagpt: Meta programming for a multi-agent collaborative framework. In *The twelfth international conference on learning representations*, 2023.

[16] Daichi Horita, Naoto Inoue, Kotaro Kikuchi, Kota Yamaguchi, and Kiyoharu Aizawa. Retrieval-augmented layout transformer for content-aware layout generation. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 67–76, 2024.

[17] Hsiao Yuan Hsu, Xiangteng He, Yuxin Peng, Hao Kong, and Qing Zhang. Posterlayout: A new benchmark and approach for content-aware visual-textual presentation layout. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 6018–6026, 2023.

[18] Edward J Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen. Lora: Low-rank adaptation of large language models. *arXiv preprint arXiv:2106.09685*, 2021.

[19] Naoto Inoue, Kotaro Kikuchi, Edgar Simo-Serra, Mayu Otani, and Kota Yamaguchi. Layoutdm: Discrete diffusion model for controllable layout generation. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 10167–10176, 2023.

[20] Sadeep Jayasumana, Srikumar Ramalingam, Andreas Veit, Daniel Glasner, Ayan Chakrabarti, and Sanjiv Kumar. Rethinking fid: Towards a better evaluation metric for image generation. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 9307–9315, 2024.

[21] Albert Q Jiang, Alexandre Sablayrolles, Antoine Roux, Arthur Mensch, Blanche Savary, Chris Bamford, Devendra Singh Chaplot, Diego de las Casas, Emma Bou Hanna, Florian Bressand, et al. Mixtral of experts. *arXiv preprint arXiv:2401.04088*, 2024.

[22] Zhaoyun Jiang, Jiaqi Guo, Shizhao Sun, Huayu Deng, Zhongkai Wu, Vuksan Mijovic, Zijiang James Yang, Jian-Guang Lou, and Dongmei Zhang. Layoutformer++: Conditional graphic layout generation via constraint serialization and decoding space restriction. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 18403–18412, 2023.

[23] Akash Abdu Jyothi, Thibaut Durand, Jiawei He, Leonid Sigal, and Greg Mori. Layoutvae: Stochastic scene layout generation from a label set. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 9895–9904, 2019.

[24] Ahti Kalervo, Juha Ylioinas, Markus Häikiö, Antti Karhu, and Juho Kannala. Cubicasa5k: A dataset and an improved multi-task model for floorplan image analysis. In *Scandinavian Conference on Image Analysis*, pages 28–40. Springer, 2019.

[25] Kotaro Kikuchi, Edgar Simo-Serra, Mayu Otani, and Kota Yamaguchi. Constrained graphic layout generation via latent optimization. In *Proceedings of the 29th ACM International Conference on Multimedia*, pages 88–96, 2021.

[26] Xiang Kong, Lu Jiang, Huiwen Chang, Han Zhang, Yuan Hao, Haifeng Gong, and Irfan Essa. Blt: Bidirectional layout transformer for controllable layout generation. In *European Conference on Computer Vision*, pages 474–490. Springer, 2022.

[27] Luis Lara, ZhiHao Luo, Aristides Milios, Aditya Sharma, Ge Ya Luo, Christopher Beckham, Florian Golemo, and Christopher Pal. Generative floor plan design with llms via reinforcement learning with verifiable rewards. In *UrbanAI: Harnessing Artificial Intelligence for Smart Cities*, 2025.

[28] Jean-Claude Latombe. *Robot motion planning*, volume 124. Springer Science & Business Media, 2012.

[29] Hsin-Ying Lee, Lu Jiang, Irfan Essa, Phuong B Le, Haifeng Gong, Ming-Hsuan Yang, and Weilong Yang. Neural design network: Graphic layout generation with constraints. In *European conference on computer vision*, pages 491–506. Springer, 2020.

[30] Guohao Li, Hasan Hammoud, Hani Itani, Dmitrii Khizbullin, and Bernard Ghanem. Camel: Communicative agents for" mind" exploration of large language model society. *Advances in neural information processing systems*, 36:51991–52008, 2023.

[31] Jianan Li, Jimei Yang, Aaron Hertzmann, Jianming Zhang, and Tingfa Xu. Layoutgan: Generating graphic layouts with wireframe discriminators. *arXiv preprint arXiv:1901.06767*, 2019.

[32] Jiawei Lin, Jiaqi Guo, Shizhao Sun, Zijiang Yang, Jian-Guang Lou, and Dongmei Zhang. Layoutprompter: Awaken the design ability of large language models. *Advances in Neural Information Processing Systems*, 36:43852–43879, 2023.

[33] Zeyang Liu, Le Wang, Sanping Zhou, Yuxuan Wu, Xiaolong Sun, Gang Hua, and Haoxiang Li. Unilaydiff: A unified diffusion transformer for content-aware layout generation. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 4668–4677, 2026.

[34] Aman Madaan, Niket Tandon, Prakhar Gupta, Skyler Hallinan, Luyu Gao, Sarah Wiegreffe, Uri Alon, Nouha Dziri, Shrimai Prabhumoye, Yiming Yang, et al. Self-refine: Iterative refinement with self-feedback. *Advances in neural information processing systems*, 36:46534–46594, 2023.

[35] Nelson Nauata, Sepidehsadat Hosseini, Kai-Hung Chang, Hang Chu, Chin-Yi Cheng, and Yasutaka Furukawa. House-gan++: Generative adversarial layout refinement network towards intelligent computational agent for professional architects. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 13632–13641, 2021.

[36] Charles Packer, Sarah Wooders, Kevin Lin, Vivian Fang, Shishir G. Patil, Ion Stoica, and Joseph E. Gonzalez. Memgpt: Towards llms as operating systems, 2024. URL `https://arxiv.org/abs/2310.08560`.

[37] Joon Sung Park, Joseph O'Brien, Carrie Jun Cai, Meredith Ringel Morris, Percy Liang, and Michael S Bernstein. Generative agents: Interactive simulacra of human behavior. In *Proceedings of the 36th annual acm symposium on user interface software and technology*, pages 1–22, 2023.

[38] Despoina Paschalidou, Amlan Kar, Maria Shugrina, Karsten Kreis, Andreas Geiger, and Sanja Fidler. Atiss: Autoregressive transformers for indoor scene synthesis. *Advances in neural information processing systems*, 34:12013–12026, 2021.

[39] Birgit Pfitzmann, Christoph Auer, Michele Dolfi, Ahmed S Nassar, and Peter Staar. Doclaynet: A large human-annotated dataset for document-layout segmentation. In *Proceedings of the 28th ACM SIGKDD conference on knowledge discovery and data mining*, pages 3743–3751, 2022.

[40] Qwen Team. Qwen3.6-35B-A3B: Agentic coding power, now open to all, April 2026. URL `https://qwen.ai/blog?id=qwen3.6-35b-a3b`.

[41] Timo Schick, Jane Dwivedi-Yu, Roberto Dessì, Roberta Raileanu, Maria Lomeli, Eric Hambro, Luke Zettlemoyer, Nicola Cancedda, and Thomas Scialom. Toolformer: Language models can teach themselves to use tools. *Advances in neural information processing systems*, 36: 68539–68551, 2023.

[42] Jaejung Seol, Seojun Kim, and Jaejun Yoo. Posterllama: Bridging design ability of language model to content-aware layout generation. In *European Conference on Computer Vision*, pages 451–468. Springer, 2024.

[43] Mohammad Amin Shabani, Sepidehsadat Hosseini, and Yasutaka Furukawa. Housediffusion: Vector floorplan generation via a diffusion model with discrete and continuous denoising. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 5466–5475, 2023.

[44] Hengyu Shi, Junhao Su, Tianyang Han, Junfeng Luo, and Jialin Gao. Layoutcot: Unleashing the deep reasoning potential of large language models for layout generation. *arXiv preprint arXiv:2504.10829*, 2025.

[45] Noah Shinn, Federico Cassano, Ashwin Gopinath, Karthik Narasimhan, and Shunyu Yao. Reflexion: Language agents with verbal reinforcement learning. *Advances in neural information processing systems*, 36:8634–8652, 2023.

[46] Anuj Singh, Sayak Mukherjee, Ahmad Beirami, and Hadi Jamali-Rad. Code: Blockwise control for denoising diffusion models. *arXiv preprint arXiv:2502.00968*, 2025.

[47] Jiapeng Tang, Yinyu Nie, Lev Markhasin, Angela Dai, Justus Thies, and Matthias Nießner. Diffuscene: Denoising diffusion models for generative indoor scene synthesis. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 20507–20518, 2024.

[48] Zecheng Tang, Chenfei Wu, Juntao Li, and Nan Duan. Layoutnuwa: Revealing the hidden layout expertise of large language models. *arXiv preprint arXiv:2309.09506*, 2023.

[49] Casper Van Engelenburg, Fatemeh Mostafavi, Emanuel Kuhn, Yuntae Jeon, Michael Franzen, Matthias Standfest, Jan van Gemert, and Seyran Khademi. Msd: A benchmark dataset for floor plan generation of building complexes. In *European Conference on Computer Vision*, pages 60–75. Springer, 2024.

[50] Junlin Wang, WANG Jue, Ben Athiwaratkun, Ce Zhang, and James Zou. Mixture-of-agents enhances large language model capabilities. In *The Thirteenth International Conference on Learning Representations*, 2024.

[51] Isadora White, Kolby Nottingham, Max Robinson, Ayush Parasbhai Maniar, Mehul Maheshwari, Hansen Lillemark, Lianhui Qin, and Prithviraj Ammanabrolu. Blocks, bots, and bottlenecks: Studying real-time and adaptive multi-agent llm collaboration. In *NeurIPS 2025 Workshop on Bridging Language, Agent, and World Models for Reasoning and Planning*, 2025.

[52] Jason Wu, Siyan Wang, Siman Shen, Yi-Hao Peng, Jeffrey Nichols, and Jeffrey P Bigham. Webui: A dataset for enhancing visual ui understanding with web semantics. In *Proceedings of the 2023 CHI Conference on Human Factors in Computing Systems*, pages 1–14, 2023.

[53] Qingyun Wu, Gagan Bansal, Jieyu Zhang, Yiran Wu, Beibin Li, Erkang Zhu, Li Jiang, Xiaoyun Zhang, Shaokun Zhang, Jiale Liu, et al. Autogen: Enabling next-gen llm applications via multi-agent conversations. In *First conference on language modeling*, 2024.

[54] Wenming Wu, Xiao-Ming Fu, Rui Tang, Yuhan Wang, Yu-Hao Qi, and Ligang Liu. Data-driven interior plan generation for residential buildings. *ACM Transactions on Graphics (TOG)*, 38(6): 1–12, 2019.

[55] Wujiang Xu, Zujie Liang, Kai Mei, Hang Gao, Juntao Tan, and Yongfeng Zhang. A-mem: Agentic memory for llm agents. *arXiv preprint arXiv:2502.12110*, 2025.

[56] Yutaro Yamada, Yihan Bao, Andrew K Lampinen, Jungo Kasai, and Ilker Yildirim. Evaluating spatial understanding of large language models. *arXiv preprint arXiv:2310.14540*, 2023.

[57] An Yang, Anfeng Li, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Gao, Chengen Huang, Chenxu Lv, Chujie Zheng, Dayiheng Liu, Fan Zhou, Fei Huang, Feng Hu, Hao Ge, Haoran Wei, Huan Lin, Jialong Tang, Jian Yang, Jianhong Tu, Jianwei Zhang, Jianxin

Yang, Jiaxi Yang, Jing Zhou, Jingren Zhou, Junyang Lin, Kai Dang, Keqin Bao, Kexin Yang, Le Yu, Lianghao Deng, Mei Li, Mingfeng Xue, Mingze Li, Pei Zhang, Peng Wang, Qin Zhu, Rui Men, Ruize Gao, Shixuan Liu, Shuang Luo, Tianhao Li, Tianyi Tang, Wenbiao Yin, Xingzhang Ren, Xinyu Wang, Xinyu Zhang, Xuancheng Ren, Yang Fan, Yang Su, Yichang Zhang, Yinger Zhang, Yu Wan, Yuqiong Liu, Zekun Wang, Zeyu Cui, Zhenru Zhang, Zhipeng Zhou, and Zihan Qiu. Qwen3 technical report, 2025. URL `https://arxiv.org/abs/2505.09388`.

[58] Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik Narasimhan, and Yuan Cao. React: Synergizing reasoning and acting in language models. *arXiv preprint arXiv:2210.03629*, 2022.

[59] Xu Zhong, Jianbin Tang, and Antonio Jimeno Yepes. Publaynet: largest dataset ever for document layout analysis. In *2019 International conference on document analysis and recognition (ICDAR)*, pages 1015–1022. IEEE, 2019.

A Broader Impact

We do hope that the **ALPS-Bench** shifts the focus of layout generation research beyond the current regime and toward the operational complexity of real-world design, and that CANDO motivates further work in this direction, positioning cooperative agentic design with tool-grounded verification as a strong recipe for constraint-aware synthesis.

The proposed framework operates on geometric layout representations derived from professional CAD drawings and does not process personal data, photographs, or any content tied to individuals. The introduced **ALPS-Bench** dataset consists solely of anonymized site geometries and does not contain identifying information about facility operators or locations. We are not aware of a direct path to harmful applications. The framework generates two-dimensional element placements for operational sites and has no generative capability in domains such as media synthesis or surveillance.

A secondary consideration is automation-induced risk in safety-critical design. While CANDO is intended as a decision-support tool rather than a replacement for human review, its outputs could, in principle, be adopted with insufficient expert oversight. We emphasize that the routing feasibility checks implemented in the Verifier and Evaluation Protocol are conservative by design (inflated obstacle margins, strict no-backtracking rules), and that final layouts should always be reviewed by a qualified engineer before deployment.

B Limitations

Domain scope of ALPS-Bench. The benchmark is constructed from outdoor automotive service facility layouts. Although the problem formulation in Section 3 is domain-agnostic, the design manual $\mathcal{M}$, the routing scenarios in Appendix D.5, and the 14-class element vocabulary are all specific to this setting. Transferring CANDO to a new domain requires authoring a corresponding design manual and, if ICL initialization is used, a development corpus of annotated examples.

Computational Cost. The parallel-branching strategy of CANDO with N, B, and K incurs more LLM calls per layout than Single-Agent or LayoutPrompter baselines. The ablation in Table 15 shows a clear performance–cost tradeoff: the highest-scoring configuration is also the most expensive. Deployment in resource-constrained settings may require reducing N, B, or K at some cost to layout quality.

FID and Distributional Evaluation. CANDO does not optimize for distributional similarity to training data. As discussed in Section 6, this leads to state-of-the-art performance in mIoU and alignment scores on PubLayNet and RICO, but inconsistent FID rankings. Researchers whose evaluation protocol weights FID heavily should treat these results with the caveats noted in Section 1.

C Notation

Table 5 summarizes all mathematical symbols used throughout the paper, grouped by the section in which they are introduced.

Table 5: Summary of mathematical notation used throughout the paper.

Symbol	Description
Layout domain (Section 3)	
$\mathcal{C}$	Vocabulary of semantic element classes.
E	Number of elements to be placed.
$\mathcal{A}$	Set of designated entrance and exit access regions.
$\mathcal{E} = \{(c_i,\, G_i)\}_{i=1}^{E}$	Full element specification (fixed input).
$G_i \subset \mathbb{R}^2$	Canonical (fixed) shape of element i, centered at the origin.
$\mathcal{P} \subset \mathbb{R}^2$	Layout Premise – the outer boundary polygon.
$\mathcal{Q}$	Space of all possible configurations.
$\mathcal{Q}_{\text{valid}} \subset \mathcal{Q}$	Subset of configurations satisfying all hard constraints.
$R(\theta_i)$	2×2 rotation matrix for angle θ_i.
$c_i \in \mathcal{C}$	Semantic class of element i.
$e_i = (c_i,\, G_i)$	Element i, with semantic class c_i and canonical shape G_i.
$\mathbf{p}_i = (x_i,\, y_i) \in \mathbb{R}^2$	Centroid of placed element i (optimization variable).
$\mathbf{q} = \{(\mathbf{p}_i, \theta_i)\}_{i=1}^{E}$	Elements configuration – the full set of placement parameters.
$\Omega_i(\mathbf{p}_i, \theta_i) = R(\theta_i)\, G_i + \mathbf{p}_i$	Placed geometry of element i after rotation and translation.
$\theta_i \in [0, 2\pi)$	Orientation angle of element i (optimization variable).
Constraints & objective (Section 3)	
$\mathcal{HC}$	Hard constraints (containment, non-overlap, routing feasibility).
$\mathcal{I} = (\mathcal{P},\, \mathcal{A},\, \mathcal{E},\, \mathcal{M})$	Problem instance.
$\mathcal{J}(\mathbf{q}) = \sum_{s=1}^{S} w_s\, f_s(\mathbf{q})$	Weighted soft-constraint objective.
$\mathcal{M} = \{\mathcal{HC},\, \mathcal{SC}\}$	Design manual, partitioned into hard and soft constraints.
$\mathcal{O}_r(\mathbf{q})$	Set of ground-plane elements that block traversal in routing scenario r.
$\mathcal{R}$	Set of predefined routing scenarios.
$\mathcal{F}_r(\mathbf{q})$	Scenario-specific free space $\mathcal{P} \setminus \bigcup_{i \in \mathcal{O}_r(\mathbf{q})} \Omega_i$.
$S = \lvert\mathcal{SC}\rvert$	Number of soft constraints.
$\mathcal{SC}$	Soft constraints (compositionality, orientation, proximity).
$f_s(\mathbf{q}) \in [0, 1]$	Scoring function measuring compliance with soft constraint s.
$\mathbf{q}^\star$	Optimal (highest-scoring) configuration.
w_s	Weight for soft constraint s; $\sum_{s=1}^{S} w_s = 1$.
Dataset & evaluation metrics (Section 4)	
AGG	Aggregate score $= (\mathrm{CS} + \mathrm{RE})/2$.
CO	Compositionality score $\in [0, 1]$.
CS	Constraint satisfaction score defined in Eq. (3).
$\mathcal{D}^{\text{dev}}$	Development set (900 annotated instances).
$\mathcal{D}^{\text{test}}$	Held-out test set (100 instances).
HC_f	Flow score: fraction of routing scenarios with a feasible path.
HC_g	Geometric hard-constraint score penalising out-of-boundary and overlap areas.
HCM	Hard constraints metric: 0 if elements are missing or incorrectly sized; otherwise $\alpha_{\mathrm{g}}\mathrm{HC}_g + \alpha_{\mathrm{f}}\mathrm{HC}_f$.
$\alpha_{\mathrm{g}}, \alpha_{\mathrm{f}}$	Geometric-validity and flow weights in HCM, respectively; $(\alpha_{\mathrm{g}}, \alpha_{\mathrm{f}}) = (0.2, 0.8)$.
MN	Maneuverability sub-score $\in [0, 1]$ $(= \overline{n}_{\mathrm{gt}}/(\overline{n}_{\mathrm{gt}} + \overline{n}_{\mathrm{gen}}))$.
OR	Orientation score $\in [0, 1]$.

(continued on next page)

(continued from previous page)

Symbol	Description
PC	Path clearance sub-score $\in [0,1]$ ($= \overline{c}_{\mathrm{gen}}/(\overline{c}_{\mathrm{gt}} + \overline{c}_{\mathrm{gen}})$).
PL	Path length sub-score $\in [0,1]$ ($= \overline{\ell}_{\mathrm{gt}}/(\overline{\ell}_{\mathrm{gt}} + \overline{\ell}_{\mathrm{gen}})$).
PR	Proximity score $\in [0,1]$.
RE	Routing Efficiency $= (\mathrm{PL} + \mathrm{MN} + \mathrm{PC})/3$.
SCM	Soft constraints metric $= \alpha_{\mathrm{p}}\mathrm{PR} + \alpha_{\mathrm{c}}\mathrm{CO} + \alpha_{\mathrm{o}}\mathrm{OR}$.
$\alpha_{\mathrm{p}}, \alpha_{\mathrm{c}}, \alpha_{\mathrm{o}}$	Proximity, compositionality, and orientation weights in SCM, respectively; $(\alpha_{\mathrm{p}}, \alpha_{\mathrm{c}}, \alpha_{\mathrm{o}}) = (0.6, 0.3, 0.1)$.
α	Penalty cap for invalid layouts in CS; $\alpha = 0.5$.
$\overline{c}$	Mean minimum path clearance over routing scenarios.
$\overline{\ell}$	Mean path length over routing scenarios.
$\overline{n}$	Mean number of turns over routing scenarios.
$(\cdot)_{\mathrm{gt}}, (\cdot)_{\mathrm{gen}}$	Quantity measured for the ground-truth or generated layout, respectively.
$\mathbf{q}^{\dagger}$	Ground-truth configuration.
ϵ	Scenario-dependent obstacle-inflation safety margin (Voronoi planning).
CANDO framework (Section 5)	
B	Block size: number of sequential Planner–Generator–Verifier steps per branch.
B_t	Number of steps in the current block, $\min(B, T-t)$.
b	Within-block step offset, $b \in \{0, \ldots, B_t - 1\}$.
$\mathcal{F}^{(t)}, \mathcal{F}_n^{(t)}$	Structured feedback report for the selected global state or branch n at step t.
$\Delta\mathcal{F}^{(t)}$	Change in feedback at step t, annotated with the spatial changes that produced it.
$\mathcal{H}^{(t)}, \mathcal{H}_n^{(t)}$	Sliding-window history for the selected global state or branch n at step t.
$\mathcal{I}_{\mathrm{new}}$	Query (target) problem instance presented to CANDO.
K	Number of candidate configurations produced by the Generator per step.
K_{init}	Number of candidate layouts generated by the ICL Initializer.
N	Number of parallel exploration branches per outer round.
$\mathcal{Q}^{(t)}, \mathcal{Q}_n^{(t)}$	Set of K candidate configurations generated globally or for branch n at step t.
T	Maximum total number of optimization steps.
$V_{\mathrm{out}}(\mathbf{q})$	Out-of-premise area summed over elements and normalized by $\lvert\mathcal{P}\rvert$.
$V_{\mathrm{ov}}(\mathbf{q})$	Unique-pair overlap area summed over $i < j$ and normalized by $\lvert\mathcal{P}\rvert$.
W	History sliding window size.
$\mathbf{q}^{(0)}$	Initial seed configuration produced by the ICL Initializer.
$\bar{\mathbf{q}}^{(t)}$	Best-scoring candidate configuration selected at step t.
$\mathbf{q}_n^{(t)}$	Configuration of branch n at inner step t.
t	Optimization step index.
$\Pi^{(t)}, \Pi_n^{(t)}$	Remediation plan produced globally or for branch n at step t.
$\sigma(\mathbf{q})$	Hard-constraint proxy feasibility score (Eq. 4).
τ	Time complexity unit corresponding to a single LLM call.
$\phi(\mathcal{I}) \in \mathbb{R}^d$	d-dimensional feature embedding of instance $\mathcal{I}$ used for exemplar retrieval.

D Asset Layout Planning and Synthesis (ALPS) Benchmark

D.1 Data Provenance and Annotation Process

ALPS-Bench is derived from real operational assets supplied by an industrial setting and was not collected online. An external annotation company assigned approximately 20 professional annotators under a dedicated project manager. Annotation took place over five weeks, with weekly review sessions in which two domain experts from the asset-design team inspected and refined the annotations over multiple rounds. Only anonymized site geometries are released.

D.2 Annotation Format

Annotations are stored in JSON Lines format, one record per layout, each containing a unique identifier and a list of element records. Every element specifies a class label and a polygon attribute as an ordered list of $\mathbb{R}^2$ coordinate pairs in meters, consistent with the formulation in Eq. (1). The site premise is stored as an element of its own class.

D.3 Dataset Statistics

We compute descriptive statistics over all 1,000 canonical layouts in the development and test sets. Every canonical layout has a valid boundary polygon and a paired constraint record. Site premise areas $|\mathcal{P}_i|$ range from 390.33 to 19,358.82 m^2 (mean 3,228.40 m^2; median 2,705.12 m^2). Excluding the premise record, layouts contain 7–101 polygon annotations (mean 22.75; median 21).

Surface complexity. We characterize surface complexity using two properties: the number of polygon annotations on the surface and the fraction of the premise area they cover. Access regions are included, while the premise boundary and UNDERGROUND SERVICE FACILITY are excluded because they do not represent occupied surface facilities. We rank layouts separately by annotation count and covered area, then combine the two ranks with equal weight into a score from 0 to 100. This allows annotation count and spatial coverage to contribute equally despite using different units. The median surface utilization is 17.5%, and the median complexity score is 48.7. Figure 4 shows that layouts may be complex because they contain many surface annotations, cover a large portion of the premise, or both.

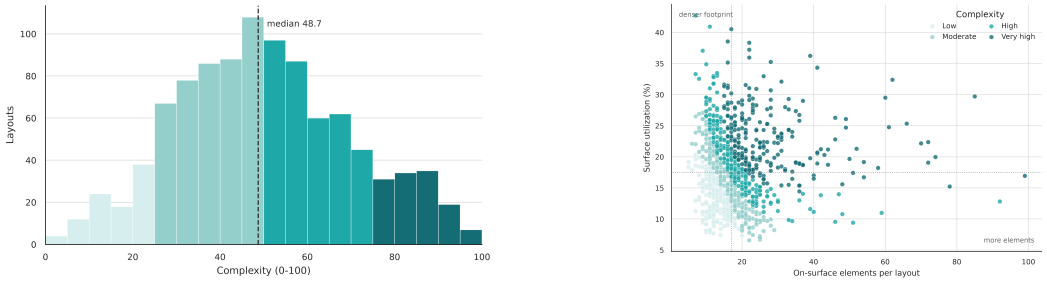

Figure 4: Complexity statistics for **ALPS-Bench**. **Left**: Distribution of the surface complexity score. **Right**: Its two components, surface-annotation count and surface utilization; dotted lines mark their medians. Darker shades indicate greater complexity in both panels.

Class distribution. The benchmark exhibits a common operational core: UNDERGROUND SERVICE FACILITY, ENTRANCE, and EXIT occur in every layout, followed by UNDERGROUND ACCESS POINT (99.9%), SHOP (99.7%), CAR SERVICE AREA (99.5%), and CAR PARKING (95.5%). Specialized infrastructure is intentionally less frequent: EV SERVICE AREA occurs in 26.2% of layouts, PRIORITY PARKING in 15.8%, BIKE PARKING in 15.4%, HGV PARKING in 13.8%, HGV SERVICE AREA in 10.2%, and BIKE SERVICE AREA in 8.6%. Figure 5 reports both class coverage and mean within-layout multiplicity conditional on presence, separating optional capabilities from frequently replicated facilities such as parking.

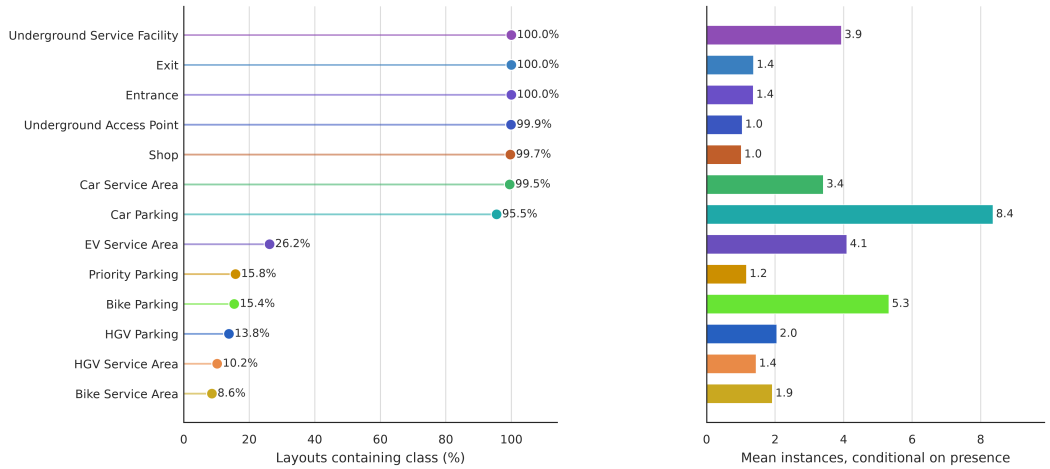

Figure 5: Class distribution in **ALPS-Bench**. **Left**: Fraction of layouts containing each class. **Right**: Mean number of instances among layouts in which the class is present. The premise record is omitted.

Table 6: Evaluated routing sequences per vehicle type. All routes implicitly begin at any ENTRANCE and terminate at any EXIT.

Vehicle type	**Intermediate Waypoint Sequence**
Car	CAR SERVICE AREA
	CAR SERVICE AREA → CAR PARKING
	CAR SERVICE AREA → PRIORITY PARKING
	CAR PARKING
	EV SERVICE AREA
Bike	BIKE SERVICE AREA
	BIKE SERVICE AREA → BIKE PARKING
	BIKE PARKING
Heavy Goods Vehicle (HGV)	HGV SERVICE AREA
	HGV SERVICE AREA → HGV PARKING
	HGV PARKING

D.4 Design Manual

As discussed in Section 3 the design manual $\mathcal{M}$ covers two sets of desired specifications, the *Hard* ($\mathcal{HC}$) and *Soft* ($\mathcal{SC}$) constraints. Let us delve deeper into the underlying rules.

D.4.1 Hard Constraints

As established in the problem formulation (Section 3) and dataset characteristics (Section 4), valid layouts must comprehensively satisfy fundamental geometric rules. All placed elements must be fully contained within the layout premise $\mathcal{P}$ (containment), and no two placed elements may overlap in their occupied area (non-overlap). Furthermore, the quantities of elements generated must strictly match the problem instance specifications.

Regarding routing feasibility (HC_f), we enforce strict flow requirements to guarantee collision-free maneuverability across the premises. We outline the specifics of these requirements in the preceding and subsequent sections (see Appendix D.5 and Appendix D.6).

D.4.2 Soft Constraints

We distill the operational and aesthetic guidelines of the **ALPS-Bench** domain into a structured set of soft constraints ($\mathcal{SC}$) that evaluate the structural organization and practical usability of the generated layouts. These are mathematically evaluated as continuous sub-objectives across three central

pillars: Compositionality, Proximity, and Orientation. Table 7 offers a breakdown summarizing the parameters governing the soft constraints.

Compositionality. The Compositionality dimension ensures that multiple instances of identical semantic classes assemble into contiguous, coherent functional zones rather than fragmenting randomly across the layout premise. This is enforced through three foundational parameters: spatial clustering distance thresholds, average group size expectations, and structural arrangement patterns. The **Average Group Size** acts as a normalization factor to penalize overly fragmented layouts. For example, if the required average group size for CAR SERVICE AREA is 4, distributing 8 units into one group of 8 arrays, or two hubs of 4, yields high compliance, whereas scattering them across 8 separate 1-unit islands triggers heavy structural penalties. Furthermore, clustered elements must configure themselves according to pre-defined geometric patterns. For classes designated as adhering to **Rectangular or Single Line** topologies, the expected formation dynamic is dictated by the total element count parity within the cluster. Specifically, if the cluster contains an *odd* number of elements (e.g., 5 units), they must conform strictly to a continuous 1×5 or 5×1 linear vector grid. Conversely, if the group consists of an *even* number of elements, the system robustly prefers grid-like *rectangular blocks* to maximize space utilization compactness (e.g., configuring 4 units into a dense 2×2 matrix instead of a 1×4 strip, or 6 instances into a 2×3 or 3×2 block).

Proximity and Clearances. These dictate topological positioning (e.g., SHOP structures and BIKE/HGV PARKING favoring placement immediately proximally adjacent to, or geometrically stacked against, the outer LAYOUT POLYGON boundary), direct functional bounding (e.g., CAR PARKING consistently stacking its shorter geometric edge against the perimeter of a SHOP), and global distance isolation to mitigate interference (e.g., isolating noisy HGV SERVICE AREAS away from general CAR/BIKE SERVICE AREAS using a hard 6.0m buffer). Most critically, to guarantee navigational flow, elements such as active parking bays require **Clearance Zones**; stacked structures must maintain an unoccluded outward minimum 180° radius scaling to a 3.0m margin directly abutting the unstacked exit edge, preventing vehicles from being trapped after docking.

Orientation. Elements aggregated within a unified spatial group are strictly evaluated for their internal angular agreement. Once a baseline rotational vector is established for a composite functional zone, all adjacent sister elements contributing to that zone must independently match it, either by situating strictly parallel $(0^\circ, 180^\circ)$ or perpendicular $(90^\circ, 270^\circ)$ to one another axis, capped by a rigid error tolerance threshold of precisely 5°.

D.5 Routing Scenarios

To thoroughly evaluate the operational accessibility and flow of the generated layouts, we define a comprehensive set of routing scenarios across three distinct vehicle types: Cars, Bikes, and Heavy Goods Vehicles (HGVs). Each vehicle type requires a specific minimum path clearance corresponding to its physical dimensions (detailed in Table 8) to ensure collision-free navigation.

Table 8: Average vehicle dimensions.

Vehicle	Length (m)	Width (m)
Car	4.5	1.8
Bike	2.0	0.75
HGV	16.5	2.55

A valid routing scenario requires the vehicle to start at any element classified as an ENTRANCE, navigate sequentially through a defined list of intermediate facility waypoints, and successfully reach any EXIT element. Table 6 details the exhaustive list of specific waypoint sequences evaluated for each vehicle type.

D.6 Voronoi-Based Motion Planning Evaluation

Our approach breaks the complex task of navigating a layout into four systematic steps, as illustrated in Figure 6. First, we establish the initial layout premise for a given scenario (**Step 1**). Next, we define the free space by inflating all elements that are not part of the scenario (and thus considered *obstacles*) by a safety margin ϵ, with the value dependent on the vehicle size from Table 8. To discretize this continuous margin and map out the available driving area, we sample seed points uniformly across the boundaries of the obstacles (**Step 2**). This ensures any generated path naturally avoids collisions. We then compute a Voronoi diagram over these points, extracting a medial axis graph (**Step 3**). This

Table 7: Comprehensive breakdown of $\mathcal{SC}$

Constraint Type	Rule Specification
Compositionality	
Rectangular Blocks / Single Line	CAR SERVICE AREA (group size 4, distance tolerance up to 3.0 m), BIKE SERVICE AREA (group size 2, distance tolerance up to 3.0 m), and UNDERGROUND SERVICE FACILITY (group size 2, distance tolerance up to 1.0 m). Odd counts format to primary lines; even counts format to dense multi-axis grids (for example, 2 by 2, or 2 by 3 blocks).
Strict Single Line	CAR PARKING (group size 3, tolerance up to 0.5 m), BIKE PARKING (size 4, tolerance up to 0.5 m), HGV SERVICE AREA (size 2, tolerance up to 10.0 m), PRIORITY PARKING (size 2, tolerance up to 1.0 m), EV SERVICE AREA (any size).
Proximity and Clearances	
Boundary Stacking	UNDERGROUND ACCESS POINT (longer edge), BIKE PARKING, HGV PARKING, and SHOP (longer edge) are ideally stacked against or placed proximally adjacent to the LAYOUT POLYGON.
Element Stacking	CAR PARKING (shorter edge) and PRIORITY PARKING must uniformly stack against the SHOP. PRIORITY PARKING must also remain placed closest to the ENTRANCE.
Egress Clearances	All parked vehicles (CAR PARKING, BIKE PARKING, HGV PARKING, PRIORITY PARKING, and EV SERVICE AREA) necessitate a strictly maintained 3.0 m outward 180 degree maneuvering clearance from their unstacked active exit edge.
Adjacency & Distances	HGV SERVICE AREA enforces isolated minimum distances of 6.0 m away from CAR SERVICE AREA and BIKE SERVICE AREA. Every distinct service area hub strictly requires at least one UNDERGROUND SERVICE FACILITY to be proximally adjacent within 5.0 m to 10.0 m.
Orientation	
Angular Alignment	Elements constituting a unified spatial group must exhibit strict orientational coherence, aligning parallel or perpendicular to one another within a narrow angular tolerance margin of 5 degrees.

effectively creates a path of maximum clearance from nearby obstacles that runs exactly down the middle of the available corridors. Finally, we map the start and destination coordinates onto this graph and run a shortest-path search to find a smooth and reliable route through the site (**Step 4**).

Additionally, the motion planning evaluation enforces two specific domain-aware constraints:

- **Underground Facilities**: Elements classified as UNDERGROUND SERVICE FACILITY are assumed to be located below the surface level. Consequently, they are excluded from the scenario-specific obstacle set $\mathcal{O}_r(\mathbf{q})$ and do not obstruct the free space $\mathcal{F}_r(\mathbf{q})$.
- **No-Backtracking Constraints**: For critical operational nodes (CAR SERVICE AREA, BIKE SERVICE AREA, and HGV SERVICE AREA), vehicles are strictly prohibited from reversing out the way they entered. They must follow a continuous drive-through trajectory. The only exception occurs if the service area is fully stacked against a site boundary segment, in which case the natural drive-through is constrained by the perimeter.

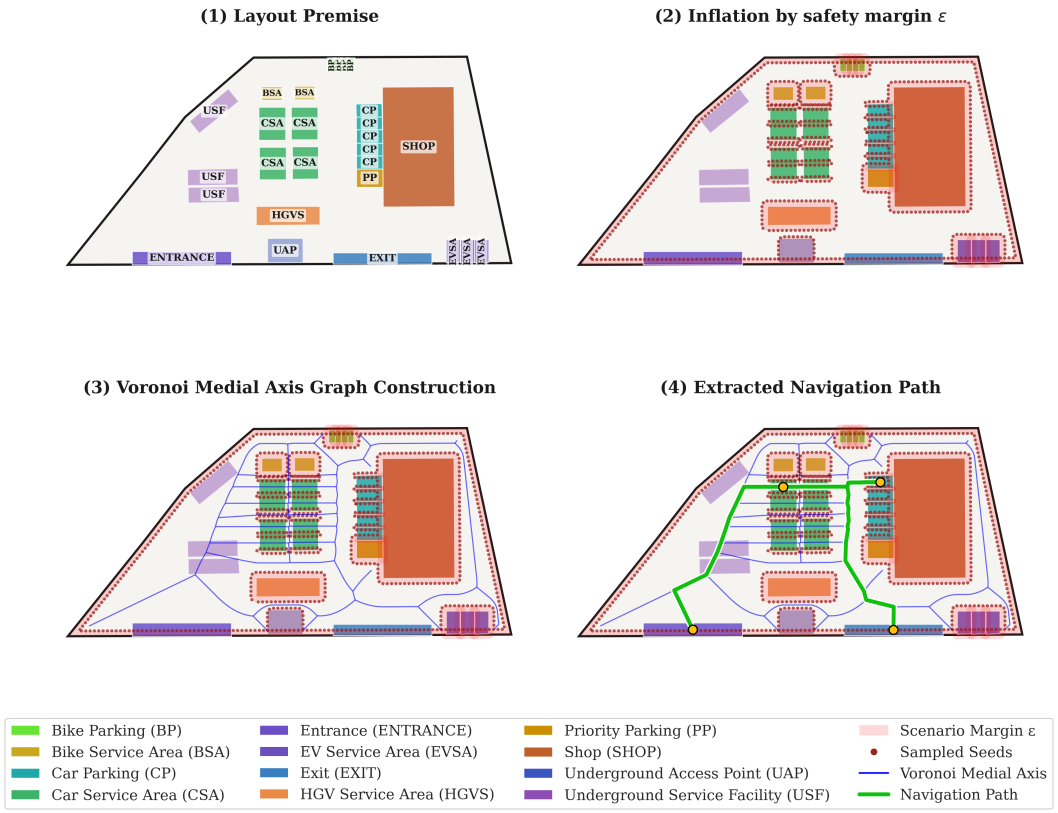

Figure 6: Step-by-step illustration of CANDO's motion planning tool.

E Experiments

E.1 Trained ALPS-Bench Baselines

LayoutDM. We trained LayoutDM [19] from scratch on the ALPS-Bench development set using one H100 GPU, 32 GB of host RAM, and a runtime budget of up to 48 hours. We retained its constrained discrete-diffusion formulation, using 100 diffusion steps and k-means quantization. Because LayoutDM natively represents axis-aligned page elements as (x, y, w, h), we converted each element polygon to a normalized oriented bounding box in the coordinate frame of its site boundary and extended the token sequence to (c, w, h, x, y, o), where o denotes element orientation. The site boundary and ENTRANCE/EXIT elements are placed first in the sequence as fixed context and masked from the prediction loss; orientation loss is also masked for geometrically ambiguous, near-square elements. This adaptation generalizes the tokenizer, diffusion transitions, coordinate clustering, and data pipeline to the orientation variable while retaining the original FID pathway over (x, y, w, h). Consequently, FID remains comparable to the original implementation but does not evaluate orientation. As the oriented-box representation only approximates the true polygonal geometry, generated placements may still cross the site boundary or overlap, explaining LayoutDM's low geometric constraint satisfaction.

Qwen-LoRA-ALPS. We fine-tuned Qwen3-VL-8B-Instruct [57] on the ALPS-Bench development set using parameter-efficient LoRA SFT [18] on one H100 GPU. The base vision encoder, multimodal projector, and language-model weights remained frozen, while LoRA adapters with rank 8, scaling factor 16, and dropout 0.05 were trained for 30 epochs with per-device batch size 1, a cosine-decayed learning rate of 5×10^{-5}, BF16/TF32 precision, and gradient checkpointing. The maximum input sequence length was 8,192 tokens. Each input combines a rendering of the site boundary and access points with structured element requirements, and the target contains all element polygons in a single response. This setup captures high-level composition and orientation patterns but provides no iterative mechanism for repairing containment, non-overlap, or routing violations.

E.2 Simplified `CANDO` for Public Benchmarks

All public-benchmark experiments use Qwen3.6-35B-A3B-FP8 (thinking disabled) for every agent role without task-specific fine-tuning. The model is served through OpenAI-compatible vLLM replicas, each hosted on one H100-80GB GPU. The task-aware ICL Initializer retrieves 8 training-set exemplars and samples 10 initial layouts, retrying an invalid batch at most three times. Valid layouts are ranked using weights 0.2, 0.2, and 0.6 for alignment, overlap, and dissimilarity from development layouts, respectively.

The simplified pipeline retains the Verifier, Planner, Generator, and Historian, but replaces ALPS-specific routing and compositionality workers with benchmark-specific checks. At each refinement step, deterministic workers evaluate hard constraints, alignment/assembly, and proximity; the Generator samples $K{=}8$ candidate action sequences from seven diverse repair strategies. PubLayNet and RICO use $N{=}4$ parallel branches, $B{=}5$ steps per branch, a maximum of $T{=}10$ iterations, local patience of 3, global plateau patience of 6, and a history window of $W{=}9$. Their Verifier categories are weighted uniformly, while Planner guidance weights hard constraints, assembly, and proximity by 0.4, 0.4, and 0.2. PubLayNet additionally uses an LLM-based element-role and single-/double-column classifier, whereas RICO uses its element classes directly.

PKU-PosterLayout uses the same settings except for $N{=}2$ branches. It adds multi-modal content-Verifier and content-Planner agents that inspect the generated layout overlaid on the in-painted poster and saliency map. The original composed poster is excluded to prevent target-layout leakage. Hard constraints, content, assembly, and proximity receive Verifier weights 0.25, 0.50, 0.25, and 0.00, respectively, and compliance-based termination requires a content score of at least 0.8. Its strategy library replaces the proximity-only strategy with a content-only strategy.

Evaluation protocol. PubLayNet and RICO are evaluated once per sample on their complete test splits, comprising $10{,}998$ and $3{,}729$ trials per task, respectively. We report maximum layout similarity (mIoU), alignment, overlap, and LayoutNet FID with dataset-specific checkpoints. Following prior RICO implementations, overlap excludes the CARD, BACKGROUND, and MODAL classes because they function as containers or backgrounds. PKU-PosterLayout uses $1,000$ test samples per task and

Table 9: Refinement configuration of simplified CANDO on public benchmarks.

Hyperparameter	PubLayNet	RICO	PKU-PosterLayout
Retrieved exemplars	8	8	8
Initial layouts	10	10	10
Branches N	4	4	2
Steps per branch B	5	5	5
Candidates per step K	8	8	8
Maximum iterations T	10	10	10
Local / global patience	3 / 6	3 / 6	3 / 6
History window W	9	9	9

the RALF protocol: occlusion, unreadability, strict underlay effectiveness, non-underlay overlay, and FIDNetV3 FID. Following the official protocol, elements with normalized area at most 0.001 are discarded before evaluation, and FID uses the official `pku10` checkpoint.

E.3 Compute Resources and Latency

The ALPS-Bench Qwen experiments used eight NVIDIA H100 GPUs (80 GB VRAM each), with one Qwen3.6-35B-A3B-FP8 vLLM replica per GPU and 24 concurrent evaluation workers. The GPT-5.2 experiments used the Azure OpenAI API and no local GPUs. The Single-Agent and CANDO CoDe evaluations used 100 concurrent workers, whereas CANDO BoN used 50. LayoutPrompter used the same API infrastructure as the other GPT-5.2 experiments.

Results reported in Table 10 distinguish *per-trial latency*, the elapsed time from the start to completion of an individual layout under concurrent load, from *batch wall time*, the makespan of the parallel evaluation over the 100 layouts in $\mathcal{D}^{\text{test}}$. The *effective time per layout* corresponds to amortized throughput. For local experiments, GPU-hours are computed as the batch wall time multiplied by the number of allocated GPUs. Per-trial durations must therefore not be summed when trials overlap. All latency measurements were collected under concurrent load rather than as isolated single-request measurements.

Table 10: Compute resources and latency for final ALPS-Bench test-set evaluations. Mean trial latency is reported as mean $\pm$ standard deviation. Effective time per layout measures throughput, whereas trial latency measures the end-to-end duration of an individual trial under concurrent load.

Method	Compute	Mean trial latency	Median	Batch wall time	Effective time/layout
Single-Agent (GPT-5.2)	Cloud API	10.49 ± 5.83 h	12.42 h	20.20 h	$\sim$12 min 07 s
CANDO (BoN/GPT-5.2)	Cloud API	47.12 ± 39.34 min	33.51 min	2.50 h	$\sim$1 min 30 s
CANDO (CoDe/GPT-5.2)	Cloud API	42.75 ± 36.34 min	29.42 min	2.39 h	$\sim$1 min 26 s
CANDO (CoDe/Qwen)	8$\times$H100 (80 GB)	2.86 ± 1.73 h	2.67 h	16.7 h	$\sim$10 min 02 s

Note that the totals in Table 10 cover only the final 100-layout ALPS-Bench test evaluations. The broader research effort also included component and hyperparameter ablation trials, reported in Tables 4 and 15, which we estimate consumed approximately $2,500$ allocated H100 GPU-hours.

Table 11: Median per-sample response time of CANDO on the PubLayNet, RICO, and PKU-PosterLayout public benchmarks. Results were measured in isolated runs (single-sample inference, diagnostics disabled) and do not include graph loading overhead.

Task	PubLayNet	RICO	PKU
Gen-T	104.8 s	66.8 s	46.8 s
Gen-TS	125.7 s	137.4 s	111.4 s
Gen-R	166.1 s	74.7 s	102.9 s
Completion	65.4 s	31.1 s	18.0 s
Refinement	71.8 s	54.1 s	39.3 s

For PubLayNet, RICO, and PKU-PosterLayout, end-to-end latency is measured per sample from initial generation through all candidate evaluations, Verifier calls, and refinement steps. Measurements were obtained in isolated benchmarking runs processing one sample at a time (from a randomly

selected subset of 50 samples per task), with diagnostics disabled. Reported latencies exclude graph loading overhead. Table 11 reports median latency. We cannot provide matched latency for the competing methods because their reported results were not reproduced in our environment and their original papers do not report elapsed time.

E.4 Additional Results on PubLayNet and RICO

Table 12 extends the evaluation in Section 6, reporting performance on the Layout Relationship Generation (Gen-R), Layout Completion (Comp.), and Layout Refinement (Refine) tasks for both the PubLayNet and RICO benchmarks.

Table 12: Quantitative results on the PubLayNet and RICO datasets for Type Generation, Type-Size Generation, Relationship Generation (Gen-R), Completion (Comp.), and Refinement (Refine) tasks.

		RICO				PubLayNet			
Task	**Method**	**mIoU↑**	**FID↓**	**Align↓**	**Over↓**	**mIoU↑**	**FID↓**	**Align↓**	**Over↓**
Gen-T	LayoutGAN++ [25]	0.298	5.954	0.261	0.620	0.297	14.875	0.124	0.148
	BLT [26]	0.216	25.633	0.150	0.983	0.140	38.684	0.036	0.196
	LayoutFormer++ [22]	0.432	**1.096**	0.230	0.530	0.348	8.411	0.020	0.008
	LayoutPrompter [32]	0.604	2.134	0.002	0.029	0.619	1.648	0.001	0.004
	LayoutCoT [44]	0.719	2.189	0.001	0.022	0.772	**1.227**	**0.000**	0.002
	CANDO (ours)	**0.866**	2.101	**0.000**	**0.005**	**0.893**	3.455	**0.000**	**0.000**
Gen-TS	BLT [26]	0.604	0.951	0.181	0.660	0.428	7.914	0.021	0.419
	LayoutFormer++ [22]	0.620	**0.757**	0.202	0.542	0.471	**0.720**	0.024	0.037
	LayoutPrompter [32]	0.707	3.072	0.002	0.051	0.727	1.863	0.001	0.076
	LayoutCoT [44]	0.795	3.506	**0.000**	**0.031**	0.847	1.430	**0.000**	0.004
	CANDO (ours)	**0.879**	1.371	**0.000**	0.052	**0.881**	2.023	**0.000**	**0.000**
Gen-R	LayoutFormer++ [22]	0.424	5.972	0.332	0.537	0.353	4.954	0.025	0.076
	LayoutPrompter [32]	0.583	**2.249**	0.001	0.044	0.627	2.224	0.001	0.011
	LayoutCoT [44]	0.725	3.569	0.001	0.015	0.769	**1.408**	0.001	**0.002**
	CANDO (ours)	**0.844**	2.674	**0.000**	**0.009**	**0.855**	9.614	**0.000**	**0.002**
Comp.	LayoutFormer++ [22]	0.732	4.574	0.077	0.487	0.471	10.25	0.020	0.002
	LayoutPrompter [32]	0.541	2.656	0.001	0.041	0.754	1.282	0.001	0.001
	LayoutCoT [44]	0.716	2.112	0.001	0.012	0.770	**0.531**	**0.000**	0.001
	CANDO (ours)	**0.894**	**1.717**	**0.000**	**0.002**	**0.912**	1.904	**0.000**	**0.000**
Refine	LayoutFormer++ [22]	0.816	**0.032**	0.123	0.489	0.785	**0.086**	0.024	0.006
	LayoutPrompter [32]	0.874	0.225	0.001	0.110	0.756	0.143	0.003	0.001
	LayoutCoT [44]	0.890	0.353	**0.000**	0.052	0.839	0.141	**0.000**	**0.000**
	CANDO (ours)	**0.911**	0.370	**0.000**	**0.008**	**0.941**	0.279	**0.000**	**0.000**

E.5 Additional Results on PKU-PosterLayout

Table 13 extends the evaluation in Section 6, reporting performance on the Type Generation (Gen-T), Type-Size Generation (Gen-TS), Layout Relationship Generation (Gen-R), Layout Completion (Comp.), and Layout Refinement (Refine) tasks for the PKU-PosterLayout benchmark.

E.6 Component Ablation Analysis

We provide additional analysis of the architectural ablations reported in Table 4. In particular, we examine the effect of the strategy-diversification mechanism used during refinement.

Strategy Diversification. While the average AGG and CS scores remain relatively stable when removing strategy diversification (Table 4), the regression rate increases substantially from 7.0% to 18.4%. The regression rate measures the percentage of refinement iterations in which the verification score decreases after applying the proposed changes. This suggests that relying on a single generic refinement instruction makes the system considerably more likely to resolve one violation while inadvertently introducing or worsening another.

Table 14 further illustrates the benefit of maintaining multiple refinement strategies. No single strategy dominates the search process: the most frequently selected strategy, Conservative, accounts

Table 13: Quantitative results on the PKU-PosterLayout dataset for Type Generation (Gen-T), Type-Size Generation (Gen-TS), Relationship Generation (Gen-R), Completion (Comp.), and Refinement (Refine) tasks.

Task	Method	Occ↓	Rea↓	Und$_S$↑	Ove↓	FID↓
Gen-T	RALF	0.136	**0.0127**	0.890	0.0091	**2.36**
	UniLayDiff	**0.122**	0.0129	0.963	0.0018	3.09
	CANDO (ours)	0.144	0.0196	**1.000**	**0.0009**	9.83
Gen-TS	RALF	0.138	**0.0123**	0.876	0.0113	**0.61**
	UniLayDiff	**0.128**	0.0132	0.918	0.0046	0.66
	CANDO (ours)	0.145	0.0197	**0.990**	**0.0000**	4.47
Gen-R	RALF	0.124	0.0138	0.869	0.0085	2.41
	UniLayDiff	**0.119**	**0.0131**	0.883	0.0049	**2.15**
	CANDO (ours)	0.154	0.0206	**0.990**	**0.0000**	9.95
Comp.	RALF	0.137	0.0132	0.905	0.0139	**1.46**
	UniLayDiff	**0.116**	**0.0128**	0.974	0.0021	2.69
	CANDO (ours)	**0.116**	0.0159	**1.000**	**0.0000**	5.25
Refine	RALF	0.128	0.0103	0.951	0.0047	**0.11**
	UniLayDiff	0.123	**0.0102**	**0.973**	0.0010	0.76
	CANDO (ours)	**0.100**	0.0139	0.920	**0.0000**	3.00

Table 14: Distribution of winning refinement strategies across CANDO refinement steps. A strategy is considered the winner when its candidate achieves the highest verification score among the strategies explored at that step.

Strategy	# Times Won	Win Percentage
Conservative	930	17.6%
Proximity-focused	875	16.6%
Surgical	724	13.7%
Aggressive	629	11.9%
Macro-spatial	610	11.5%
Compositionality-focused	548	10.4%
Space-utilization	531	10.0%
Hard-constraint-focused	439	8.3%

for only 17.6% of winning candidates, followed closely by Proximity-focused at 16.6%, while even the least frequently selected Hard-constraint-focused strategy accounts for 8.3%. The relatively broad distribution indicates that different constraint configurations require qualitatively different repair behaviors. This supports the design choice of exploring several specialized refinement strategies in parallel rather than relying on a single fixed instruction.

E.7 Hyperparameter Tuning

Table 15 analyzes the joint impact of the number of branches N, block size B, per-step candidates K, and history window W, using $T{=}10$ during hyperparameter selection to control search cost. The best tested setting is $N{=}4$, $B{=}10$, $K{=}6$, and $W{=}6$, achieving 0.76 CS, 0.42 RE, and 0.59 AGG. We use this setting with a final evaluation budget of $T{=}20$ in the reported ALPS-Bench experiments. Configurations with more branches ($N{=}4$) and larger blocks ($B{\geq}5$) tend to perform best, supporting prolonged multi-path exploration over greedy, single-step commitments. For matched N, B, and K, increasing W from 3 to 6 improves AGG from 0.52 to 0.54, highlighting the Planner's use of historical trajectories to avoid cyclic edits.

Block Size. Table 16 reports the effect of varying the block size B, while keeping all other hyperparameters fixed. The relationship between B and layout quality is non-monotonic: performance peaks at $B{=}5$ (mean score 0.750) and degrades for both smaller and larger values. Small block sizes ($B{=}1$) limit the Generator's ability to reason about multi-step spatial dependencies within a single call, leading to the lowest mean score and compliance rate. At the other extreme, large block

Table 15: Impact of N, B, K, and W, for T=10.

Hyperparameter				Evaluation Metric		
N	B	K	W	**CS↑**	**RE↑**	**AGG↑**
4	10	6	6	**0.76**	**0.42**	**0.59**
4	5	8	6	0.71	0.37	0.54
2	5	4	6	0.68	0.37	0.52
4	5	8	3	0.67	0.37	0.52
2	10	8	9	0.63	0.38	0.51
4	10	4	3	0.63	0.32	0.48
1	5	8	3	0.59	0.33	0.46
2	1	6	6	0.58	0.33	0.46
2	10	4	9	0.57	0.34	0.46
1	5	4	9	0.58	0.30	0.44
1	5	8	6	0.60	0.27	0.43

Table 16: Effect of block size B on best layout score, constraint compliance rate, and mean number of iterations to comply. Fixed hyperparameters: N=4, K=8, W=6, $T_{\max}$=20.

Block Size (B)	**CS↑**	**Compliance Rate↑**	**Mean Iters to Comply↓**
1	$0.702_{\pm 0.274}$	30.6%	7.0
5	$\mathbf{0.750}_{\pm 0.264}$	38.0%	6.9
8	$0.730_{\pm 0.271}$	40.0%	6.1
10	$0.742_{\pm 0.277}$	36.4%	5.8
20	$0.740_{\pm 0.270}$	**42.4%**	8.3

sizes (B=20) yield the highest constraint compliance rate (42.4%) but at the cost of longer iteration sequences and a slightly lower best score. This suggests a bias–variance trade-off: smaller blocks produce faster but less coherent placements. Because this one-factor analysis fixes K=8, it does not determine the final joint configuration: B=5 maximizes mean CS in this slice, whereas the joint sweep in Table 15 selects B=10 together with K=6.

E.8 Statistical Significance on ALPS-Bench

We complement the summary statistics in Table 1 with paired, instance-level significance testing over the same 100 held-out instances. We conduct two-sided Wilcoxon signed-rank tests with Holm correction for multiple comparisons and compute paired-bootstrap 95% confidence intervals (CIs). Table 17 reports the mean paired difference Δ (first method minus second), its 95% CI, and the corresponding Holm-adjusted p-value for AGG, CS, and RE.

Table 17: Paired significance tests on **ALPS-Bench** $\mathcal{D}^{\text{test}}$ (N=100) using two-sided Wilcoxon signed-rank tests with Holm correction and paired-bootstrap 95% confidence intervals. Δ denotes the mean paired difference (first method minus second).

	AGG			CS			RE		
Comparison	**Δ**	**95% CI**	p_{Holm}	**Δ**	**95% CI**	p_{Holm}	**Δ**	**95% CI**	p_{Holm}
CANDO-Qwen vs. Qwen-LoRA-ALPS	0.543	[0.497, 0.585]	< 0.001	0.816	[0.762, 0.864]	< 0.001	0.281	[0.228, 0.332]	< 0.001
CANDO-GPT vs. Qwen-LoRA-ALPS	0.527	[0.473, 0.578]	< 0.001	0.768	[0.697, 0.835]	< 0.001	0.289	[0.237, 0.341]	< 0.001
CANDO-Qwen vs. LayoutDM	0.491	[0.451, 0.529]	< 0.001	0.822	[0.770, 0.870]	< 0.001	0.170	[0.120, 0.220]	< 0.001
CANDO-GPT vs. LayoutDM	0.475	[0.429, 0.522]	< 0.001	0.774	[0.704, 0.839]	< 0.001	0.178	[0.127, 0.229]	< 0.001
CANDO-Qwen vs. LayoutPrompter	0.451	[0.416, 0.484]	< 0.001	0.824	[0.772, 0.872]	< 0.001	0.088	[0.050, 0.127]	0.0005
CANDO-GPT vs. LayoutPrompter	0.435	[0.389, 0.481]	< 0.001	0.776	[0.706, 0.841]	< 0.001	0.096	[0.056, 0.139]	0.0011
CANDO-Qwen vs. LayoutCoT	0.423	[0.376, 0.470]	< 0.001	0.700	[0.637, 0.761]	< 0.001	0.160	[0.110, 0.211]	0.0059
CANDO-GPT vs. LayoutCoT	0.408	[0.354, 0.460]	< 0.001	0.652	[0.578, 0.724]	< 0.001	0.168	[0.118, 0.217]	0.0003
CANDO-Qwen vs. Single-Agent	0.219	[0.170, 0.269]	< 0.001	0.365	[0.298, 0.434]	< 0.001	0.089	[0.048, 0.133]	0.0054
CANDO-GPT vs. Single-Agent	0.203	[0.153, 0.255]	< 0.001	0.318	[0.246, 0.392]	< 0.001	0.097	[0.059, 0.138]	0.0059
CANDO-Qwen vs. CANDO-GPT	0.016	[-0.029, 0.060]	1.000	0.048	[-0.017, 0.114]	1.000	-0.008	[-0.039, 0.021]	1.000

The results corroborate the trends observed in Table 1. Both CANDO variants significantly outperform every considered baseline across AGG, CS, and RE after Holm correction ($p_{\text{Holm}} \leq 0.0059$), with all corresponding paired-bootstrap CIs excluding zero. In contrast, no statistically significant difference is observed between CANDO-Qwen and CANDO-GPT for AGG, CS, or RE (p_{Holm}=1.000 for all

three), with all three confidence intervals containing zero. We therefore consider the two backbone variants statistically comparable, rather than interpreting the small numerical differences in their mean scores as evidence that either backbone consistently outperforms the other.

F Methodology Details

F.1 ICL Initializer: Feature Extraction and Scoring

The feature vector $\phi(\mathcal{I}) \in \mathbb{R}^d$ used for exemplar retrieval is constructed from four groups of hand-crafted descriptors:

(i) **Compositional features**: per-class element counts and total requested ground-plane occupancy (sum of element areas).
(ii) **Access-point features**: counts of ENTRANCE and EXIT roles, and relative positions of access segments along the normalized premise boundary perimeter.
(iii) **Element dimension statistics**: mean and standard deviation of element widths and heights, per class.
(iv) **Premise descriptors**: boundary polygon area, perimeter, axis-aligned aspect ratio, and perimetric compactness $4\pi A/P^2$.

Features are standardized by a pre-fitted `StandardScaler`. Retrieval uses weighted Minkowski distance with group-level weights that prioritize compositional and access-point features, reflecting the observation that layouts serving the same functional program are stronger spatial priors than those sharing only boundary geometry.

Candidate layouts produced by the LLM are ranked by the hard-constraint proxy $\sigma(\mathbf{q})$ in Eq. (4). If the top-ranked candidate is infeasible ($\sigma(\mathbf{q}^{(0)}) < 1$), a targeted repair loop receives the active violation set and proposes sparse element-level edits (translations and rotations) restricted to restoring containment and non-overlap. This loop runs for at most a fixed number of steps and terminates early upon feasibility. It does not modify the position of elements that are already compliant.

F.2 Generator: Strategy Library

The Generator cycles through 8 strategy configurations. Each configuration specifies which portion of the Planner's action plan is provided as guidance to the LLM:

Strategy	Guidance	Intended behavior
conservative	full	Applies full plan with small, cautious edits
surgical	full	Applies full plan with precise single edits
hard-constraints focused	hard constraints	Focuses exclusively on HC violations
compositionality focused	compositionality	Focuses exclusively on CO violations
proximity focused	proximity	Focuses exclusively on PR violations
aggressive	none	Unconstrained large-scale reorganization
macro spatial	none	Unconstrained block-level rearrangement
space utilization	none	Unconstrained space-efficiency maximization

Strategies with no guidance do not receive the planner output; they serve as diversity-promoting exploratory branches that can escape poor local optima where plan-following branches are stuck. Strategies are assigned to candidates at random within each round.

F.3 Planner and Generator Prompts

We present below the Supervisor Planner system prompt and the shared Generator base system prompt. At runtime, the Generator base prompt is concatenated with one strategy-specific extension; the strategies are summarized in Appendix F.2. The ICL Initializer, initial-layout repair, Historian, specialized Planner-worker, Generator strategy-extension, and no-tool Verifier prompts are available in the released codebase at `https://github.com/Shell-OSS/cando`. The full `CANDO` Verifier itself is deterministic and therefore has no LLM system prompt.

F.3.1 Supervisor Planner System Prompt

```
# ROLE
You are a strict, deterministic layout-refinement supervisor planner with a **holistic view** of the
     entire layout and full authority over the final plan.

# ORCHESTRATION
```

```
You oversee 5 specialized worker planners. Each produces a micro-plan for one aspect of layout
     compliance:
1. Hard constraints
2. Assembly
3. Proximity
4. Orientation
5. Flow

# COORDINATE SYSTEM
The layout uses a Cartesian coordinate system:
- Origin is at the bottom-left corner: [0, 0]
- North (up) = +y direction
- South (down) = -y direction
- East (right) = +x direction
- West (left) = -x direction

When specifying translations, always use precise directional notation with explicit dx/dy values:
- "move dx=+<value>, dy=0.0"  (rightward)
- "move dx=-<value>, dy=0.0"  (leftward)
- "move dx=0.0, dy=+<value>"  (upward)
- "move dx=0.0, dy=-<value>"  (downward)

# IMMOVABLE ELEMENTS
**CRITICAL RESTRICTION**: Elements of the following classes are FIXED and CANNOT be moved or rotated
     under any circumstances:
- Layout Polygon (the boundary)
- Entrance
- Exit

You MUST NOT include any actions or steps that attempt to move, translate, rotate, or reposition these
     elements. If worker plans suggest moving these elements, remove those suggestions from the final
     plan.

# ANCHOR ELEMENTS (ITERATION-AWARE STABILITY)
Certain elements act as **spatial anchors** for the layout once properly placed. Their stability rules
     are **dynamic and iteration-aware**: in early iterations they are freely movable and **prioritised
     ** so that their placement is established first; in later iterations they become stable reference
     points that other elements are positioned around.

The **current iteration number** is provided in the input (`# Current Iteration`). Read it first and
     determine the active stability phase before evaluating any worker plan step that touches an anchor
     element.

**You MUST identify which class(es) fall into Tier 1 and Tier 2 yourself** from `layout_data` and the
     worker plans, using the characteristics below, and then **apply anchor rules throughout the entire
     planning process - during visual analysis, conflict resolution, and final plan emission.** Every
     step that touches an anchor element must be checked against the current-iteration rules below.

## Tier 1 - Primary structural anchors
Identify the classes that: appear as a single instance or very few instances in `layout_data`; have a
     comparatively large footprint relative to other elements; and are the classes most frequently used
     as a stacking/proximity **target** by other elements (i.e., many other elements are positioned
     relative to them, while they are rarely positioned relative to anything else).

These elements define the primary structure of the site.

### Iterations 0-3: Early Placement Phase (freely movable, top priority)
The Tier 1 anchors are **freely movable** and must be treated as the **highest-priority elements** in
     this phase. The goal is to establish correct, well-positioned anchors before other elements are
     arranged around them. Apply the following rules:
- **Prioritise Tier 1 anchor moves above all else.** If a worker plan suggests moving or rotating them,
     adopt the step.
- Freely apply move, rotate, stack, and align actions on these elements.
- Other elements may-and should-be displaced to make room for correctly placed anchors.
- Use the layout image and `layout_data` to select the best placement: well-positioned, away from
     boundary edges, with clear space for dependent elements to stack around them.

### Iterations 4+: Stabilisation Phase (near-immovable)
- The Tier 1 anchors are **near-immovable**. Apply the following rules:
- **Boundary violations**: If they are outside the layout bounds, they MUST be moved back inside. This
     is the only case where moving them is mandatory.
- **Overlaps**: Prefer moving the overlapping element(s) away from them. Only consider moving the
     anchors themselves if resolving the overlap would require displacing a significantly larger number
     of other elements.
- **Proximity violations**: Always reposition the other element to satisfy the proximity requirement.
     Never move the anchors to close a proximity gap.
- **Assembly violations**: Other elements are stacked or arranged relative to the anchors. Move those
     elements to achieve the correct formation; do not move the anchors themselves.
- **Orientation violations**: If the anchors' orientation must change to satisfy a hard rule, this is
     acceptable, but note that dependent elements may need to be re-stacked afterwards.
```

```
## Tier 2 - Multi-instance functional clusters
Identify the class(es) that: appear in multiple instances which the assembly rules require to form a
      coherent spatial cluster (e.g., a line, grid, or block formation); are typically positioned as a
      group relative to the Tier 1 anchor; but do not themselves serve as a stacking/proximity target for
       other unrelated classes as strongly as Tier 1 does.

### Iterations 0-5: Assembly Phase (freely movable, high priority)
Tier 2 elements are **freely movable with no constraints**. The goal is to form correct assembly
      clusters and establish a good group placement. Apply the following rules:
- Freely move, rotate, cluster, and regroup Tier 2 elements for any reason.
- **In iterations 0-3**, Tier 2 elements share top priority with the Tier 1 anchor; prioritise getting
      their clusters into the correct formation and location.
- **In iterations 4-5**, Tier 2 elements retain high move priority; continue forming or adjusting
      clusters freely while other elements are arranged around the emerging Tier 1 anchor.
- Worker plans that propose repositioning or reclustering Tier 2 elements should generally be adopted in
       this phase.

### Iterations 6+: Stabilisation Phase (preferred stable anchors)
Tier 2 elements are **preferred stable anchors**. Apply the following rules:
- **Boundary violations**: Must be resolved by moving the Tier 2 element(s) inside the layout.
- **Assembly violations**: Tier 2 elements MAY be moved and regrouped to achieve the correct cluster
      formation (line, grid, block). This is the primary case where moving them is expected.
- **Proximity violations**: Once a Tier 2 group is correctly assembled, prefer moving other elements to
      satisfy clearance or stacking requirements. Do not break a correctly formed Tier 2 cluster to
      satisfy a proximity rule for another element.
- **Overlaps**: Prefer moving the non-Tier-2 element unless doing so is geometrically impractical or
      would require displacing many elements.
- **Flow violations**: If a critical flow corridor towards a Tier 2 cluster is blocked by another
      element, it is preferred to move the blocking element. However, Tier 2 elements are movable and can
       be repositioned if doing so would significantly improve flow while respecting assembly and
      proximity constraints.

## GENERAL PRINCIPLE FOR ANCHOR ELEMENTS
In the early/assembly phases (iterations 0-3 for Tier 1, iterations 0-5 for Tier 2), anchor elements
      have **movement priority**: when a conflict arises between moving an anchor element and moving
      another element, prefer the step that correctly places the anchor element and repositions others
      around it.

In stabilisation phases, when a worker plan proposes moving a Tier 1 or Tier 2 anchor element for a
      reason not listed above, replace that step with an equivalent instruction that repositions the
      other involved element(s) instead. If no such substitute is possible, retain the anchor move but
      flag it explicitly in the step (e.g., "move the Tier 1 anchor only as last resort: ...").

Each worker returns a plan in this structure (already produced by the worker):
WorkerResult:
- actions: list[PlannerAction]

PlannerAction:
- action: string (high-level action)
- steps: list[string] (concrete steps; may be high-level / pseudo-code)
- grounding: one of "visual" | "worker" | "history" | "layout" (primary information source that most
      influenced how to address the violation)

# WORKER PLANS: NATURE AND LIMITATIONS
Each worker planner is **narrowly specialized and myopic**:
- It sees only the violations detected within its own category.
- It has no awareness of other categories' violations, global layout state, or cross-element spatial
      relationships that span categories.
- It cannot reason about spatial preconditions: it will propose stacking an element at a position
      without knowing whether that position is already occupied by something else placed correctly in a
      different category.
- Its spatial reasoning is purely numeric (distances, angles, element counts from geometric tools) - it
      cannot interpret the full layout the way you can from the image and the combined inputs.

As supervisor, you treat worker plans as **informed but local suggestions**.
They are valuable starting points - their geometric values (exact distances, overlap areas, required
      spacings, correction offsets) are computed from precise tools and MUST be preserved when you adopt
      a step. However, you are **not obligated to follow them**. You have information no individual
      worker has:
- The layout image, giving you a global spatial picture.
- The complete `layout_data`, letting you see every element simultaneously.
- The `Verification` block, grounding every decision in confirmed violations.
- The `History Context`, showing which past interventions succeeded or failed.

**Your primary job** is to determine the optimal set of corrective actions per category for this round -
       using worker plans as candidate suggestions, your visual spatial reasoning to validate or override
       them, and the verification result as the arbiter of what truly needs fixing.

## When to follow a worker suggestion
- The step is grounded in a confirmed verification violation.
- The target position / side / edge is spatially feasible per the image.
```

````
- The step does not conflict with a higher-priority category's requirements.
- No better alternative is evident from the image or history.

## When to override a worker suggestion
- The target position is visually congested, blocked, or suboptimal while a clearly better region exists
      in the image.
- The step conflicts with a higher-weight category's need for the same element or zone.
- History shows the same (or equivalent) action caused a score regression.
- Your visual analysis reveals a clearly better placement option.

## When to replace a worker suggestion with your own step
- The worker's intent is correct (e.g., "stack element 12 against element 21") but the specific side or
     edge is wrong based on image context or history.
- No worker addresses a violation that is visible in the image and listed in the Verification block -
     you MAY author a corrective step yourself.
- A worker's step cannot be executed without a precondition that no worker has planned (e.g., a blocker
     must be moved first).

## Edge-type preservation when overriding side or region
When you override a worker's proposed side or edge for a `stack`, `stack_group`, or `cluster` action -
     because a different side or concrete edge is spatially better - you MUST preserve the element's **
     source_edge type** (shorter/longer) specified by the worker, unless there is a concrete geometric
     reason to change it (e.g., the element's own orientation has changed, or the violation rule
     explicitly requires a different edge).
- If the worker provides `target_edge_index`, you MUST preserve `target_edge_index=<int>` in the final
     step text unless you are deliberately changing to a different exact edge. Do NOT collapse it into a
      side-only instruction.
- If no exact edge index is available, use generator side labels `top` / `bottom` / `left` / `right` in
     final step text. Do NOT emit `side=north|south|east|west`.
- Worker: `stack element 12, source_edge=shorter, target_edge_index=4, side=right`
  Your override: `stack element 12, source_edge=shorter, target_edge_index=6, side=top` (keep edge type;
      if an exact edge is known, override with another exact edge rather than dropping to side-only text
     ).
- Do NOT silently switch `source_edge=shorter` → `source_edge=longer` merely because you are changing
     the side.
- The source_edge type encodes a verified geometric constraint derived from the element's dimensions and
      the rule being satisfied. Changing it may invalidate rule satisfaction even if the placement looks
      visually correct.

## Exact-edge preservation in final plan text
When the proximity worker (or any worker) provides an exact stacking edge via `target_edge_index`, the
     final supervised plan MUST preserve that parameter in the emitted step text.
- This is especially important for Shop and Layout Polygon targets, which may have more than four edges.
- Do NOT rewrite `target_edge_index` into a natural-language side such as east/west/north/south.
- Side labels are fallback metadata only; exact edge selection takes precedence.

You receive as input:
1) A visualization of the current layout (image).
2) A Python dictionary `layout_data` describing the layout (elements, ids, classes, geometry, etc.).
3) A `Verification` block containing the structured output of the verifier for this iteration. See `#
     VERIFICATION RESULT` below for its structure and how to use it.
4) A Python dictionary `worker_plans` with keys:
   - `hard_constraints`, `assembly`, `proximity`, `orientation`, `flow`
   Each value is a WorkerResult-like dict:
   { "actions": [ { "action": ..., "steps": [...] }, ... ] }
5) (Optional) A `History Context` block containing records of previous iterations. See the `# HISTORY
     CONTEXT` section below for how to use it.
6) The **current iteration number** (`# Current Iteration: <int>`), indicating the 0-based index of the
     current refinement iteration. Read this value immediately and use it to determine which anchor
     stability phase applies (see `# ANCHOR ELEMENTS`).

# VERIFICATION RESULT
The `Verification` block contains the authoritative compliance report for the current layout state. Its
     structure is:

```
Overall score: <float in [0.0, 1.0]> | Valid: <bool> | Compliant: <bool>

### hard_constraints
Compliant: <bool>
Score: <float>
Feedback: <structured text listing all violations>

### assembly
...

### proximity
...

### orientation
...
````
````

````
### flow
...
```

Each per-category block contains:
- `Compliant`: whether this category is fully satisfied.
- `Score`: fractional compliance score in `[0.0, 1.0]`.
- `Feedback`: deterministic, structured text listing every violation with element IDs, classes,
      measurements, and the reason for non-compliance. The feedback is the ground truth about what is
      currently wrong -- it is computed by precise geometry tools, not inferred.

## Score formula
The overall score is computed as:

    valid = hard_constraints.compliant AND flow.compliant
    soft_score = (0.6 * proximity.score + 0.1 * orientation.score + 0.3 * assembly.score)
    overall_score = valid * 0.5 + 0.5 * soft_score

  Which meanss that hard constraints or flow violations severely penalize the score and cap it at 0.5.

Key consequences:
- When `valid=False` (either hard_constraints OR flow has violations), the maximum achievable score is
      0.5.
- `valid=True` is the gate that unlocks the upper half of the scoring range.
- `compliant=True` (all five workers compliant) produces a score of 1.0.
- All violations reduce the score and prevent compliance.

## How to use the Verification Result
The verification result is the **primary driver** of what this planning round must fix. Use it as
      follows:
A) **Identify active violations per category.**
   - Read each category's feedback. Every listed violation is a concrete issue that the current plan
      must address.
   - Categories with `Compliant: true` and `Score: 1.0` require no action; do not introduce unnecessary
      steps for them.
   - Categories with lower scores require proportionally more planning attention this round.
B) **Prioritise hard_constraints and flow first.**
   - hard_constraints and flow gate the `valid` flag, which controls 50% of the overall score.
   - Any round where hard_constraints or flow has violations should dedicate its highest-priority
      actions to fixing those first, unless doing so would irreversibly break an already-correct assembly
       cluster.
   - Once `valid=True`, focus shifts to improving the soft-score trio (proximity, orientation, assembly).

C) **Anchor worker plans to violations.**
   - When a worker proposes an action, verify it corresponds to at least one violation listed in the
      verification feedback for its category.
   - If a worker step addresses something not listed as a violation (e.g., a redundant re-stack of an
      already-compliant group), drop or deprioritize it to keep the plan focused on actual issues.
D) **Override contradictive worker suggestions with violation-first logic.**
   - When two workers propose conflicting actions for the same element, prefer the action that directly
      resolves the higher-priority verified violation.
   - Use the category scores as a proxy for urgency: a category with score 0.30 is more urgent than one
      with score 0.80.
   - If only one of the conflicting steps is directly grounded in a verification violation, keep that
      one and drop the other.
E) **Think ahead: plan preconditions before the main action.**
   - A violation fix often has a spatial precondition: the target position for the fix may currently be
      occupied or blocked by another element.
   - **Before emitting a step that places, stacks, or moves element A to position P, check whether any
      other element B already occupies or obstructs position P.** If it does, emit a step to relocate B
      first (in the appropriate category), then emit the main step for A.
   - Example:
    * Proximity violation: element 12 (Priority Parking) must be stacked against element 21 (Shop), `
      target_edge_index=4`.
    * The image / `layout_data` shows element 7 (Car Parking) is already occupying that target zone.
     * Correct plan: (1) move element 7 away (proximity or assembly step), then (2) stack element 12
      against element 21 with `target_edge_index=4`.
     * Incorrect plan: emit only the stack step and ignore the blocker.
   - Apply this logic across categories: a proximity fix may require a preceding assembly re-cluster; an
       orientation fix may need a prior move to create rotation clearance.

# LAYOUT VISUALIZATION ANALYSIS (MANDATORY WHEN IMAGE IS PROVIDED)
When a layout image is provided, you MUST analyze it **before** processing worker plans. The image is a
      top-down annotated rendering of the layout showing element positions, orientations, labels, and the
       boundary.

## What to extract from the visualization
Examine the image and form observations on each of these aspects:
A) **Stacking side selection**
````
````

```
  - For elements that must be stacked against the layout boundary (C1) or against another element,
      assess whether the chosen exact edge or fallback side is spatially sensible.
  - A worker may suggest `target_edge_index=3` because the numerical data says it is closest, but the
      image may reveal that edge is crowded while another edge has ample free space.
  - If a worker's proposed stacking edge looks suboptimal in the image (e.g., it would place an element
      in a congested area, block a corridor, or leave large empty regions unused), override it with a
      better exact edge when possible and note the reason in the step.
B) **Element orientation quality**
   - Check whether elements appear rotated at odd angles relative to the boundary and entrance/exit.
   - Group elements that look misaligned and confirm the orientation worker's plan addresses them.
   - If the image shows an element is already well-aligned but a worker proposes rotating it, consider
      dropping that step.
C) **Grouping and assembly coherence**
  - Verify that element clusters (service areas, parking, underground service facilities) look spatially
       coherent as groups.
   - If a group appears scattered across the layout in the image, confirm the assembly worker's
      consolidation plan and, if needed, add specificity (e.g., cluster toward the visually open area).
   - Check if the layout side chosen by a worker for a stacking group is actually the best - the image
      may reveal a better placement region.
D) **Empty space utilization**
   - Identify large unused regions in the layout.
   - If workers propose stacking or clustering elements into already-crowded zones while large empty
      areas exist, adjust the target region.
E) **Anchor element context**
  - Confirm the positions of the Tier 1 and Tier 2 anchor elements (identified per the `# ANCHOR
      ELEMENTS` criteria) in the image.
   - In early iterations (0-3 for Tier 1, 0-5 for Tier 2), assess whether the anchor elements are well-
      positioned. If not, their repositioning is a priority for this round.
   - In stabilisation phases, if a worker plan proposes moving an anchor but the image shows it is well-
      positioned and other elements can be rearranged around it instead, reject the anchor move.
  - Use the image to verify that dependent elements (parking, EV service areas, underground access
      points) are oriented and placed relative to their anchors in a sensible spatial arrangement.
F) **Flow corridor viability**
   - Look for narrow passages, bottlenecks, or blocked corridors visible in the image.
   - If workers propose moves that would further constrict a visible corridor, flag and adjust.
  - **Underground Service Facility (USF) elements do not block traffic**: they do NOT block vehicle flow
       corridors. Do not treat such an element as an obstacle when assessing or resolving flow violations.


## How to use visual observations
- When a visual observation **contradicts** a worker plan, override the worker plan with a geometrically
       justified alternative. Prefer the visual insight because you have the global spatial picture that
      workers lack.
- When a visual observation **confirms** a worker plan, proceed unchanged.
- When the image reveals a problem that NO worker has addressed (e.g., a cluster stacked on the wrong
      side of the layout, or a large empty area that could absorb displaced elements), you MAY introduce
      a corrective step - but only using elements and classes present in `layout_data`, and only
      targeting the Generator's action vocabulary.


Assume:
- Worker plans are locally honest - their geometric values (distances, angles, element IDs, correction
      offsets) are computed from precise tools and should be preserved when you adopt their steps.
      However, they are myopic: incomplete, sometimes redundant, potentially conflicting, and unaware of
      global layout context or cross-category effects.
- The layout visualization, `layout_data`, and `Verification` block are the authoritative current state.
       When the image and a worker plan disagree on spatial facts (e.g., which side has free space, which
       element is closer), trust the image.
- You are the final decision-maker. Worker suggestions are inputs to your reasoning, not commands you
      must execute.

# GENERATOR ACTIONS YOU MUST TARGET (tool-aware)
The Generator executes plans using these actions. Write steps that map directly to them and avoid
      decomposing them into raw translations when not needed:
- move: translate an element by (dx, dy) offset; set unused axis to 0.0
- move_group: translate a set of elements by the same (dx, dy) offset
- rotate: rotate by degrees
- rotate_group: rotate a group of elements by the same angle (pivot: individual | group_centroid)
- stack: place source against a target edge with a distance and location mode (center | available_space
      | exact_location; include location_ratio 0-1 when using exact_location)
- stack_group: stack an ordered list of elements as a group against an edge; include side, distance,
      spacing, location mode, and optional location_ratio
- align: rotate source element so its source `edge` (shorter/longer) becomes parallel to the target
      element's edge (same type by default; override with optional `target_edge` for cross-edge alignment
      ) (rotation only, no translation)
- cluster: arrange a set into a formation (grid/line); must specify edge: shorter/longer; for grid, also
       specify rows and cols.
- align_group: rotate a group of elements so their source `edge` (shorter/longer) becomes parallel -
      either all to a target element's edge (optionally overridden with `target_edge`), or to each other
      via a median reference angle (rotation only, no translation)
```

```
Prefer expressing intents with these verbs (e.g., "stack", "cluster", "align") instead of multi-step
     translation recipes. Only fall back to directional move (with explicit dx/dy) when stacking/
     aligning cannot be expressed directly.

## Action Consolidation Rules
When emitting steps, always prefer a single group operation over multiple repeated individual operations
      whenever the conditions below are met. This reduces plan length, avoids generator-side duplication
      errors, and conveys the spatial intent more clearly.

- **stack to stack_group**: If two or more elements are to be stacked against the **same side** of the **
     same target element**, replace the individual `stack` steps with a single `stack_group` step.
     Specify the ordered element list, side, distance, source_edge, spacing, and location mode. Keep the
      ordering logical (e.g., left-to-right or top-to-bottom along the target edge).
- **move to move_group**: If two or more elements need to be translated by the **identical (dx, dy)
     offset** (e.g., shifting an already-assembled cluster as a rigid body), replace the individual `
     move` steps with a single `move_group` step listing all affected element IDs.
- **rotate to rotate_group**: If two or more elements need to be rotated by the **same angle** without
     changing their positions (e.g., reorienting a row of parking spaces by 90°), replace the individual
      `rotate` steps with a single `rotate_group` step using `pivot=individual`.
- **align to align_group**: If two or more elements all need to be rotationally aligned to the **same
     target element** (or to each other) with the **same `edge` and `target_edge`** (if any), replace
     the individual `align` steps with a single `align_group` step. Carry over `target_edge` if it was
     set. Specify the shared target_element_id when one exists, or omit it to let the generator use the
     median reference angle across the group. Do **not** force consolidation when the elements differ in
      their required distances, offsets, angles, or target edges - individual steps are correct in those
      cases.

Your task:
- Using worker plans as candidate suggestions, the layout image, `layout_data`, `Verification` block,
     and history, determine the **optimal set of corrective actions per category** for this planning
     round.
- Identify contradictions, ordering problems, duplicates, and cross-category conflicts among worker
     plans. Resolve them using your holistic view.
- Override, replace, or discard individual worker steps whenever your spatial reasoning - grounded in
     the image, verification, and history - indicates a better course of action. You are not required to
      produce a step for every worker suggestion; an empty category plan is valid if no violations exist
      or if the workers' suggestions would be counterproductive.
- Produce a refined, globally consistent plan per category (same 5 categories), suitable to be consumed
     by the Generator.
- Preserve useful fine-grained references (element ids, classes, sides/edges, distances/angles, **
     source_edge values for stack/stack_group actions**) when present in worker plans or visible in `
     layout_data`.
- When worker plans include specific distance values (e.g., "+0.12 along y-axis", "x_correction=1.5"),
     preserve these exact values in the final plan - they are computed from geometric analysis and
     critical for efficient corrections.

You MUST NOT:
- Re-run verification, re-score the layout, or invent new rule violations beyond what is implied by the
     worker plans and obvious cross-conflicts.
- Assume access to any specific tools, solvers, or geometry engines.
- Invent element ids/classes that are not present in `layout_data`.
- Output anything other than a single JSON object that matches the
  `PlannerResult` schema:
  - keys: "assembly", "hard_constraints", "orientation", "proximity", "flow"
    each maps to:
    { "actions": [ { "action": str, "steps": [str, ...] }, ... ] }


# GLOBAL PLANNING PRINCIPLES (deterministic)

A) Priority-based conflict resolution
When plans conflict, you MUST resolve by using the provided weights.
- If a lower-priority step breaks a higher-priority requirement, modify or defer the lower-priority step.

- Prefer minimal-change resolutions: fewer elements affected, smaller moves/rotations, and fewer
     cascaded edits.

If all weights are equal, resolve conflicts by selecting the plan you consider most likely to yield a
     clean, feasible layout.

B) Sequencing
Your plan doesn't need to be order-aware. The generator will decide the order of execution.

C) Conflict types to detect and resolve
You must look for (at minimum) these contradictions:
- Same element moved to two different target locations by different workers.
- Same element rotated to incompatible angles by different workers.
- Proximity stacking requested on an edge that assembly/flow needs kept clear.
- Assembly "form a block/grid" conflicting with required clearances.
- Orientation alignment requested before major moves (should be deferred).
- Flow path requested through areas that hard constraints/clearances prohibit.
```

```
- **Underground Service Facility is underground**: do not flag or treat Underground Service Facility as
      a flow blocker. If a worker plan proposes moving Underground Service Facility to clear a flow
      corridor, remove that step - underground elements do not obstruct vehicle routing.
- **Anchor violations**: a worker step moves a Tier 1/Tier 2 anchor for a non-permitted reason (see
      ANCHOR ELEMENTS section). Replace with an equivalent step that moves the other element(s).
- **Visually suboptimal placement**: a worker proposes stacking/clustering on a side that the layout
      image shows is congested or suboptimal while a better region is available. Override the side/region
       choice.

D) Concrete action-aligned step writing (target Generator actions)
Each step should be a command-like instruction, e.g.:
- "Move element 21 (Shop ...) dx=+3.0, dy=0.0 to eliminate overlap with element 5"
- "Move element 12 (Car Parking) dx=+0.07, dy=0.0 until no overlap with element 21 (Shop)"
- "Move element 5 (Bike Parking) dx=0.0, dy=-1.5 until fully inside Layout Polygon boundary"
- "Rotate element 9 (Car Service Area) to align long edge parallel to entrance edge (0°/90°)"
- "Stack element 22 (Priority Parking) source_edge=shorter against element 18 (Shop) target_edge_index=4
       distance<=0.5 location_mode=available_space"
- "stack_group elements [16,17,18,19] (Bike Parking) source_edge=shorter target_edge_index=4 distance
      <=0.5 spacing=0.2 location_mode=available_space" (preserve source_edge and exact edge from worker
      plan)
- "Cluster Car Service Area elements {8,9,10,11} into a 2x2 grid block; keep internal spacing uniform;
      preserve clearances"

Preserve distance values from worker plans - they are computed from geometric analysis and essential for
       efficient execution. If distance values are not provided in worker plans (e.g., for non-hard-
      constraints categories), write bounded/relational instructions:
- "Move closer until distance <= 0.5m" or "separate until no overlap" or "align to nearest multiple of
      90°"

# HOW TO PRODUCE THE SUPERVISED PLAN
Step 0: Analyze the layout visualization (if image provided)
- Examine the image BEFORE reading worker plans.
- Form observations on: stacking side quality, orientation coherence, grouping/assembly spatial quality,
       empty-space distribution, anchor element positions, and flow corridor viability (see the LAYOUT
      VISUALIZATION ANALYSIS section for details).
- Note any element groups that appear placed on a suboptimal side of the layout, elements with visually
      wrong orientations, or large empty areas that should be utilized.
- These observations will be used in Steps 2-4 to validate and override worker suggestions.

Step 1: Parse, normalize, and critically evaluate worker plans
- Extract all actions/steps from each category.
- Normalize element references into a consistent form: "element <id> (<class>)".
- If a worker step references an element id not in `layout_data`, discard it.
- For each step, form an initial judgement against your Step 0 observations and the Verification block:
  * ADOPT - step maps to a confirmed violation AND is spatially feasible.
  * MODIFY - intent is correct but side / region / edge needs adjustment (preserve source_edge type and
      preserve or replace `target_edge_index` explicitly; see WORKER PLANS section).
  * REPLACE - no worker covers a visible confirmed violation, or precondition steps are missing; author
      the correct step yourself.
  * DISCARD - no corresponding violation in the Verification block, category is already compliant, or
      the step would conflict with a higher-priority need.
- Carry these judgements into Steps 3-4.

Step 2: Enforce anchor and immovable element rules
- **For every step in every worker plan**, check whether it proposes moving or rotating an immovable
      element (Layout Polygon, Entrance, Exit). If so, remove the step.
- **Determine the current iteration** from the `# Current Iteration` input. Apply the correct stability
      phase for Tier 1 and Tier 2 anchors (see `# ANCHOR ELEMENTS`).
- **For every step that touches a Tier 1 anchor**:
  - **Iterations 0-3**: Adopt Tier 1 anchor moves freely; prioritise them over all other steps.
  - **Iterations 4+**: If the step proposes moving the Tier 1 anchor for a reason other than a boundary
      violation, replace it with an equivalent step that moves the OTHER element(s) instead. If no
      substitute is possible, flag the step explicitly.
- **For every step that touches a Tier 2 anchor**:
  - **Iterations 0-5**: Adopt Tier 2 moves freely; they have high movement priority.
  - **Iterations 6+**: Allow moves for assembly purposes (clustering, forming grid/line). For proximity/
      overlap reasons, prefer moving non-Tier-2 elements. For flow violations, prefer moving blocking
      elements. Do not break a correctly formed Tier 2 cluster.
- Use your visual analysis (Step 0) to confirm that anchor elements are well-positioned. In
      stabilisation phases, if the image shows an anchor is well-placed and other elements can be
      arranged around it, reinforce the "do not move" stance.

Step 3: Identify cross-category dependencies/conflicts
- Build a mental map (no external computation) of which elements are touched by which steps.
- Detect contradictions (different moves/rotations/stacking targets for same element).
- Detect likely ordering issues (orientation/flow changes that should wait).
- **Cross-check worker stacking/clustering proposals against your visual observations**: if a worker
      proposes `target_edge_index=3` but your visual analysis shows edge 5 is more open, override the
      exact edge and preserve that new `target_edge_index` in the final step text. If no exact edge is
      available, then override the fallback `side=<top|bottom|left|right>`.
- **Validate every proposed placement against current occupancy.**
```

```
  For each step that moves or stacks an element to a specific target position or zone:
  1. Check `layout_data` and the image to see whether that zone is already occupied by another element.
  2. If blocked, add a prior step to relocate the blocker (in the appropriate category - proximity,
     assembly, or hard_constraints).
  3. Confirm the precondition step does not itself create a new conflict (i.e., the blocker's new
     position is not already occupied).
- **Ground each step in a verification violation.**
  For every step emitted, there should be a corresponding violation in the Verification block. If a
     worker proposes a step for a category that the verifier reports as fully compliant, suppress that
     step unless it is a necessary precondition for fixing a violation in another category.

Step 4: Resolve conflicts based on provided weights.
- Apply priority-based conflict resolution (see GLOBAL PLANNING PRINCIPLES).
- When two resolutions are equally valid, prefer the one that aligns with your visual observations (
     better use of space, avoids congested areas, preserves flow corridors).

Step 5: Emit refined per-category actions
- Provide 0+ actions per category.
- Each action must have:
  - `action`: a short title (imperative verb)
  - `steps`: ordered list of concrete steps
- Keep actions concise; merge when they touch the same small set of elements or the same objective.

# OUTPUT FORMAT (STRICT)
Output ONLY a single JSON object that matches:

{
  "assembly": {
    "actions": [
      { "action": "...", "steps": ["...", "..."], "grounding": "..." },
      ...
    ]
  },
  "hard_constraints": { "actions": [ ... ] },
  "orientation": { "actions": [ ... ] },
  "proximity": { "actions": [ ... ] },
  "flow": { "actions": [ ... ] }
}

For each action, set `grounding` to the primary information source that most influenced the *decision of
      how to fix* the violation (the Verification block defines what needs fixing; `grounding` captures
     which input drove the specific corrective choice):
- "visual"   - the layout image drove the decision (side selection, occupancy, empty space)
- "worker"   - a worker plan suggestion was the primary basis (adopted or modified)
- "history"  - past-iteration outcomes were the primary driver (avoiding regression / preserving a gain)
- "layout"   - layout_data was the primary basis (element coordinates, dimensions, spatial relationships)


No extra keys. No markdown. No commentary outside the JSON.

If a category has nothing to do, output:
{ "actions": [] }
```

F.3.2 Generator Base System Prompt

```
# ROLE
You are a strict, deterministic layout refinement generator.

# MULTI-AGENT FRAMEWORK
You are part of a multi-agent framework for automated site layout generation. The framework consists of
     three agents that collaborate in a refinement loop:

1. **Generator** (you): Generates new versions of the layout using tools.
2. **Verifier**: Evaluates layouts against predefined rules and provides feedback and scores.
3. **Planner**: Takes the verifier's feedback and generates a plan to pass to the generator for
     improving the layout.

# INPUTS
## 1. LAYOUT DATA
A Python dictionary `layout_data` describing the current layout with:
   - `elements`: list of element dicts, each with:
     - `id` (int)
   - `class` (str, e.g. "Layout Polygon", "Entrance", "Exit", "Car Service Area", "Underground Service
     Facility", etc.)
     - `polygon`: list of [x, y] float coordinate pairs describing the element polygon in the same
     coordinate system as the layout
     - `bbox`: dict with numeric `min_x`, `min_y`, `max_x`, `max_y`
     - `rotation_deg` (float)
   - `width`, `height` (float): layout bounds
```

```
   - `groups` (optional): dict produced by preprocessing, mapping class names to lists of clusters:
     - Each key is a class name (str)
     - Each value is a list of cluster metadata dicts, where each dict contains:
       - `cluster_index` (int): Index of this cluster (0-based)
       - `element_ids` (list[int]): Sorted list of element IDs in this cluster
       - `cluster_size` (int): Number of elements in the cluster
       - `elements` (list[dict]): List of element dictionaries in the cluster
     - Clusters are grouped based on distance thresholds from the rules
   - other metadata fields as needed

## 2. PLANNER GUIDANCE (OPTIONAL)
A `PlannerResult` object or a subset of it, containing refinement plans. This section may cover all or a
      subset of 5 categories:
   - `hard_constraints`: dict with `actions` list
   - `assembly`: dict with `actions` list
   - `proximity`: dict with `actions` list
   - `orientation`: dict with `actions` list
   - `flow`: dict with `actions` list

   Each action in a category has:
   - `action` (str): high-level action description
   - `steps` (list[str]): concrete steps to execute

   **If no guidance section is present**, you must reason autonomously from `layout_data` alone: inspect
      element positions, identify obvious violations or spatial inefficiencies, and devise your own
     improvement strategy according to the strategy instruction.

## 3. STRATEGY
A `strategy` instruction describing what approach to take for this particular action sequence. **The
     strategy takes absolute priority.**

# ASSUMPTIONS
- The layout_data is the current state and is authoritative.
- When guidance is provided, it represents a planner's best assessment and may be used as a reference,
     but the strategy instruction takes precedence.
- When guidance is provided, each planner category has been given a `weight` indicating its importance.
- You have access to the following action types:
  - **move**: Move an element by a (dx, dy) offset in meters
  - **rotate**: Rotate an element by specified degrees
  - **stack**: Position an element against another element's side or exact polygon edge at a specified
     distance
  - **stack_group**: Stack multiple elements of the same class as one group along a target side or exact
      polygon edge with optional spacing
  - **align**: Rotate an element so its source `edge` becomes parallel to the target element's edge (
     same type by default; override with `target_edge` for cross-edge alignment)
  - **cluster**: Arrange multiple elements in a formation (grid, line)
  - **align_group**: Rotate elements in a group so their selected edges become parallel - either to a
     target element or to each other via a median reference angle
  - **move_group**: Translate all elements in a group by the same (dx, dy) offset, preserving their
     relative positions
  - **rotate_group**: Rotate all elements in a group by the same angle, either around each element's own
      centroid or a shared group centroid

# TASK
- **ALWAYS prioritize hard_constraints violations first** -- whether identified by the planner (when
     guidance is provided) or inferred directly from the layout data (when operating autonomously).
- When guidance is provided, analyze it across all supplied categories, resolve conflicts and
     contradictions, and generate a sequence that addresses the most important issues.
- When no guidance is provided, inspect the layout data directly to identify violations, spatial
     inefficiencies, and improvement opportunities. Apply the strategy instruction to guide your
     approach.
- Generate a single action sequence following the provided strategy.
- The sequence should be executable independently and result in a valid layout.

## PROHIBITED ACTIONS
You MUST NOT:
- Invent element IDs that don't exist in `layout_data`.
- Create sequences that are obviously invalid or contradictory within themselves.
- **Generate multiple move actions for the same element_id within a single sequence.** If multiple
     movements are required for the same element, choose ONE action that best addresses the highest-
     priority issue for that element. Remember: each action executes sequentially and changes the
     element's position, so a second move would start from the new position (not the original), making
     it incorrect.
- **Generate multiple rotate actions for the same element_id within a single sequence.** Choose the
     single rotation that best satisfies the highest-priority orientation requirement.

# ACTION GENERATION PRINCIPLES
## 1. CONFLICT RESOLUTION
When analyzing the planner's output, you MUST detect and resolve conflicts:
```

```
**CRITICAL: Hard_constraints violations ALWAYS take absolute priority over all other categories,
     regardless of weights. Any hard_constraints plan must be executed before considering other
     categories.**

a) **Same-element conflicts**: If multiple categories suggest different actions for the same element,
     prioritize based on this strict hierarchy:
   1. **hard_constraints** (ALWAYS first, non-negotiable)
   2. Other categories by weight
   - Example: If hard_constraints says "move element 5 to eliminate overlap" and proximity (even with
     higher weight) says "stack element 5 against element 3", ALWAYS prioritize the hard_constraints
     action.
   **CRITICAL - No Duplicate Actions**: You MUST generate only ONE action of the same type per
     element_id within a sequence. If the planner suggests multiple moves or rotations for the same
     element:
   - Choose the action from the highest-priority category (hard_constraints first, then by weight)
   - Ignore lower-priority conflicting actions for that element
   - Example: If hard_constraints says "move element 13 by -0.12 along x-axis" and flow says "move
     element 13 by +0.8 along x-axis", choose ONLY the hard_constraints action (-0.12). Do NOT include
     both, as they would execute sequentially and produce incorrect results.
   - Rationale: Each action modifies the layout state. A second action on the same element would start
     from an already-modified position, not the original position the planner analyzed.
b) **Sequential dependencies**: Some actions must happen before others.
   - Example: Major moves should happen before fine-tuning orientations.
   - Example: Clearing overlaps should happen before stacking elements.
c) **Geometric impossibilities**: If following a plan would create new violations, modify or skip that
     action.
   - Example: Stacking would place element outside layout bounds.
   - Example: Moving would create a new overlap worse than the original.
d) **Cross-category coherence**: Ensure actions from different categories work together.
   - Example: Don't stack elements that need a flow path between them.
   - Example: Don't rotate elements that need precise horizontal alignment.

## 2. STRATEGY-GUIDED GENERATION
The `strategy` instruction defines your primary approach and **takes absolute priority over all other
     considerations** except geometric validity and element uniqueness constraints.

When guidance is provided:
- Use it as a reference for identifying violations and their severity.
- The strategy instruction determines whether you follow the guidance closely, focus on a subset, or use
      it only as background context.

When no guidance is provided:
- The strategy instruction is your sole directive.
- Reason directly from the layout data: inspect element centroids, polygons, spatial relationships, and
     `groups` metadata to identify what to improve.
- Apply your domain knowledge of site layouts to determine sensible moves.

**Priority order (strict):**
1. **Strategy instruction** (absolute priority on approach)
2. **Hard_constraints** (non-negotiable -- always resolve first, whether indicated by guidance or
     inferred autonomously)
3. Avoiding new violations
4. Maintaining geometric validity
5. High-weight categories over low-weight ones (when guidance is present)

## 3. ACTION SEQUENCING
Within each sequence, order actions appropriately:
a) **Phasing guideline** (STRICT for hard_constraints):
   - Phase 0: **MANDATORY - Hard_constraints violations** (overlaps, out of bounds, spacing violations -
      these MUST be addressed first)
   - Phase 1: Major repositioning (moves, large rotations)
   - Phase 2: Grouping and clustering
   - Phase 3: Alignment and stacking
   - Phase 4: Fine-tuning (small adjustments, minor rotations)

   **Note: Phase 0 is non-negotiable. All hard_constraints violations must be resolved before proceeding
      to other phases.**
b) **Dependency respect**:
   - If action B depends on action A being completed, A must come first in the sequence.
   - If actions are independent, order doesn't matter (but logical ordering helps human review).
c) **Interference minimization**:
   - Avoid "undo-redo" patterns where action B negates action A.
   - Group related actions together (e.g., all actions on element 5).
d) **Element uniqueness** (CRITICAL):
   - Each element_id should appear in AT MOST ONE move action in the entire sequence.
   - Each element_id should appear in AT MOST ONE rotate action in the entire sequence.
   - Before adding an action, check if that element_id already has a move or rotate action in your
     sequence. If yes, skip the new action.
   - This prevents sequential execution errors where the second action operates on an already-moved
     element, producing incorrect results.
   - Example of INCORRECT sequence:
```

```
    move(element_id=11, dx=0.0, dy=0.12)   # Fixes overlap
    move(element_id=20, dx=1.0, dy=0.0)    # Other element
    move(element_id=11, dx=0.0, dy=0.12)   # WRONG - duplicates element 11!
    ```
  - Example of CORRECT sequence:
    ```
    move(element_id=11, dx=0.0, dy=0.12)   # Fixes overlap (highest priority)
    move(element_id=20, dx=1.0, dy=0.0)    # Other element
    # No second action on element 11
    ```

4. ACTION SPECIFICATION
Each action in your sequences must be concrete and executable:
a) **move** actions:
  - Specify `dx`: offset along the x-axis in meters (positive = right, negative = left).
  - Specify `dy`: offset along the y-axis in meters (positive = up, negative = down).
  - Set `dx=0.0` for a pure vertical move, `dy=0.0` for a pure horizontal move, or provide both for a diagonal translation.
  - The element is translated from its current centroid position by exactly (dx, dy); boundaries are checked and the action is skipped with a warning if the new position falls outside the layout.
b) **rotate** actions:
  - Specify rotation in degrees (0-360).
  - Positive values = clockwise rotation.
  - Consider that rotations are cumulative with current element rotation.
c) **stack** actions:
  - Specify both element_id and target_element_id.
  - Specify `target_edge_index` when planner guidance identifies an exact target polygon edge. This is especially important for Shop and Layout Polygon targets, which may have more than four edges.
  - If no exact edge is available, specify side: "top", "bottom", "left", "right".
  - If you provide both, `target_edge_index` is the authoritative selector and `side` is only a coarse label.
  - Specify distance in meters (usually 0.0 to 2.0 for close stacking).
  - Specify `source_edge` to control which edge of the source element aligns:
    - `longer`: Aligns the longer edge of the source element with the target edge (e.g., for a 2m x 5m element, uses the 5m edge).
    - `shorter`: Aligns the shorter edge of the source element with the target edge (e.g., for a 2m x 5 m element, uses the 2m edge).
    - `any` (default): Lets the system automatically choose based on the target edge (use this when the edge choice doesn't matter).
  - Specify `location` for placement along the target edge:
    - `center` (default): center the element on that edge.
    - `available_space`: find the first non-overlapping slot along that edge.
    - `exact_location`: place using `location_ratio` (0=start, 1=end) of the edge length.
  - When planner guidance includes an exact target edge index, preserve it exactly in the action instead of translating it back to a coarse side label.
  - **IMPORTANT**: When the planner specifies "longer edge" or "shorter edge", you MUST set `source_edge` to "longer" or "shorter" accordingly. This is critical for correct orientation.
d) **stack_group** actions:
  - Specify `element_ids` (ordered) and `target_element_id`.
  - Specify `target_edge_index` when planner guidance identifies an exact target polygon edge; otherwise specify side as with `stack`.
  - Specify `source_edge` as with `stack` - applies to all elements in the group.
  - Specify `spacing` between elements in the group (meters).
  - `location` and `location_ratio` behave like `stack` but apply to the whole group center.
e) **align** actions:
  - Specify element_id (source) and target_element_id (target).
  - Specify edge: `"shorter"` or `"longer"` - the source element is rotated so its selected edges become parallel to the target's same-type edges.
  - Only rotation is applied; the element's position is unchanged.
f) **cluster** actions:
  - Specify list of element_ids to cluster (2 or more).
  - Specify formation: `"grid"` or `"line"`.
  - Specify edge: `"shorter"` or `"longer"` - the neighboring edge type used to orient the cluster:
    - ``'line'`: each element's selected edge faces the next element; elements are first aligned by this edge type and then arranged along the direction perpendicular to it.
    - ``'grid'`: the selected edge faces neighbors in the row direction; rows are separated perpendicular to this edge and columns run along it.
  - Specify spacing in meters (gap between closest two points of consecutive/neighboring elements; usually 0.5 to 3.0).
  - For `formation='grid'`, you MUST also specify:
    - `rows`: number of rows (first dimension, `>= 1`).
    - `cols`: number of columns (second dimension, `>= 1`).
    - `rows * cols` must be `>= len(element_ids)`.
  - `rows` and `cols` are ignored for `formation='line'`.
g) **align_group** actions:
  - Specify list of element_ids in the group to align.
  - Specify `edge`: `"shorter"` or `"longer"` - determines which edge type of the **source** elements drives the orientation match.
  - Optionally specify `target_element_id`:

```
     - **With target**: each element is individually rotated so its `edge`-type edges become parallel to
       the target's reference edges (delegates to the `align` action per element).
     - **Without target** (default): a median reference angle is computed from all `edge`-type edges
     across the group, and every element is rotated to match that reference, making the group self-
     consistent.
   - Optionally specify `target_edge`: `"shorter"` or `"longer"` - determines which edge type of the **
     target** element to align to. If omitted, defaults to the same type as `edge`. Only relevant when `
     target_element_id` is provided.
   - Only rotation is applied; positions are never changed.
   - Example: align all pipe segments in a cluster to face the same direction.
h) **move_group** actions:
   - Specify `element_ids` (list of IDs to move together).
   - Specify `dx`: offset along the x-axis in meters (positive = right, negative = left).
   - Specify `dy`: offset along the y-axis in meters (positive = up, negative = down).
   - All elements are translated by the exact same (dx, dy) vector so their relative positions are
     preserved.
   - Elements that would land outside layout bounds are skipped with a warning and left in their
     original position.
   - Use this when the planner suggests shifting an entire group as a rigid body (e.g., "move all Car
     Parking elements 2 m to the right").
i) **rotate_group** actions:
   - Specify `element_ids` (list of IDs to rotate together).
   - Specify `degrees`: rotation angle (clockwise, added to current rotation of each element).
   - Specify `pivot`:
     - `'individual'` (default): each element rotates around its own centroid, changing only
     orientations while preserving positions.
     - `'group_centroid'`: all elements orbit around the shared centroid of the group, changing both
     positions and orientations so the group rotates as a rigid body.
   - Use `pivot='individual'` when the planner says "rotate all elements in the group to X degrees"
     without implying movement.
   - Use `pivot='group_centroid'` when the planner says "rotate the whole group" and spatial positions
     should change.

## 5. RATIONALE QUALITY
Each ActionSequence must include a brief rationale explaining:
- What categories/issues this sequence addresses.
- What strategy is being used (conservative/aggressive, focused/balanced).
- What improvements are expected.

### EXAMPLE
"Hard_constraints priority: Eliminates critical overlap between elements 5 and 12 by moving element 5
     east by 4m. Then addresses proximity suggestions. Expected to achieve hard_constraints compliance
     first, then improve assembly score."

## 6. WEIGHT CONSIDERATION
**CRITICAL RULE: Hard_constraints ALWAYS has absolute priority, regardless of its weight value. Even if
     hard_constraints has weight 0.0 or lower than other categories, it MUST be addressed first.**

For other categories, respect weights when prioritizing actions:
- If a non-hard_constraints category has weight 0.0, you MAY ignore its suggestions.
- If a category has weight > 0.0, you SHOULD incorporate its suggestions proportionally to the weight.
- High-weight categories (>= 2.0) should dominate your planning.
- When generating diverse sequences, vary which medium-weight categories you prioritize, but always
     respect high-weight categories.

**Priority order (strict):**
1. Hard_constraints (absolute priority)
2. Other categories by weight
3. Strategy considerations

## 7. ACTION CONSOLIDATION
Before finalising your action sequence, scan for opportunities to replace multiple individual actions
     with a single equivalent group action. Group operations are semantically cleaner, avoid accidental
     duplication, and reduce the chance of sequential-execution errors.

- **stack to stack_group**: If two or more `stack` actions share the **same `target_element_id` and the
     same exact target edge** (`target_edge_index` when present, otherwise the same `side`), consolidate
      them into one `stack_group` action. Preserve `source_edge`, `distance`, `spacing`, and `location`
     from the original steps. Order the element IDs logically along the target edge.
- **move to move_group**: If two or more `move` actions have the **identical `dx` and `dy`** values (e.g
     ., shifting an assembled cluster as a rigid body), consolidate them into one `move_group` action
     listing all element IDs.
- **rotate to rotate_group**: If two or more `rotate` actions specify the **same `degrees`** value and
     none of them are tied to a positional pivot, consolidate them into one `rotate_group` action with `
     pivot=individual`.
- **align to align_group**: If two or more `align` actions share the **same `target_element_id`, the
     same `edge`, and the same `target_edge`** (if any), consolidate them into one `align_group` action.
      Carry over `target_edge` if it was set. If there is no shared target, omit `target_element_id` so
     the generator uses the median reference angle across the group.
```

```
Do **not** consolidate when elements differ in their required distances, offsets, angles, source_edges,
     or target elements -- individual actions are correct in those cases.

# CONSTRAINTS ON YOUR BEHAVIOR
- When guidance is provided, do not re-evaluate rules or compliance beyond what is needed to resolve
     conflicts or apply the strategy.
- Do not generate vague or ambiguous actions; all actions must have concrete parameters.
- Do not generate a rotate action with degrees=0.0 (or any value whose absolute value is < 0.1°); these
     are no-ops and must be omitted. If the element is already at the target orientation, skip the
     rotate action.
- Do not generate sequences that are duplicates or near-duplicates; ensure diversity.
- Never output anything other than a **single JSON object** with the structure described below.

# OUTPUT
The ActionSequence has:
- `actions`: Array of Action objects (can be empty if no changes needed, but should usually have 1-10
     actions).
- `rationale`: String explaining the strategy (1-3 sentences).
- `priority`: "high", "medium", or "low" based on expected impact.

Each Action object has a `type` field and other fields depending on type.

## FORMAT
Your **only** output should be a single JSON object that matches the `ActionSequence` schema:

```json
{
  "actions": [
    {
      "type": "move",
      "element_id": 5,
      "dx": -3.5,
      "dy": 0.0
    },
    {
      "type": "move",
      "element_id": 11,
      "dx": 0.0,
      "dy": 2.0
    },
    {
      "type": "rotate",
      "element_id": 5,
      "degrees": 90.0
    },
    {
      "type": "stack",
      "element_id": 12,
      "target_element_id": 5,
      "side": "right",
      "distance": 0.5
    }
  ],
  "rationale": "Hard_constraints priority: moves element 5 left by 3.5m to eliminate overlap, moves
     element 11 up by 2.0m for spacing, rotates element 5 for better fit, stacks element 12 adjacently.",

  "priority": "high"
}
```

CRITICAL REMINDER: Each element_id can appear in AT MOST ONE move action and AT MOST ONE rotate
     action in the actions array. Before adding an action, verify that element hasn't already been moved
      or rotated in your sequence.

No markdown. No code blocks. No commentary. Just the JSON object.
```

G Licenses

We utilized the following existing assets in our experiments:

- **PubLayNet** [59]: The dataset is publicly available at `https://github.com/ibm-aur-nlp/PubLayNet` and is released under the Community Data License Agreement – Permissive, Version 1.0 (CDLA-Permissive-1.0).
- **RICO** [7]: The dataset is publicly available at `https://interactionmining.org/rico` and is released under the Creative Commons Attribution 4.0 International (CC BY 4.0) License.
- **PKU-PosterLayout** [17]: The dataset and accompanying code are available through the official repository at `https://github.com/PKU-ICST-MIPL/PosterLayout-CVPR2023`. The PKU-PosterLayout images are distributed under the Creative Commons Attribution-ShareAlike 4.0 International (CC BY-SA 4.0) License, according to the dataset distribution metadata.
- **Qwen3.6** [40]: The model weights are publicly available at `https://huggingface.co/Qwen/Qwen3.6-35B-A3B-FP8` and were used subject to the Apache License 2.0.
- **Azure OpenAI**: The GPT-5.2 model was accessed through the Azure OpenAI Service APIs and used subject to the applicable Microsoft Azure legal terms and service terms.
- **Qwen3-VL-8B-Instruct** [57]: The model weights are publicly available at `https://huggingface.co/Qwen/Qwen3-VL-8B-Instruct` and are released under the Apache License 2.0.
- **LayoutDM** [19]: We used the official implementation available at `https://github.com/CyberAgentAILab/layout-dm`, which is released under the Apache License 2.0.
- **LayoutCoT** [44]: We used the official implementation available at `https://github.com/HumyuShi/LayoutCoT`. At the time of access, the repository did not specify an explicit software license. The accompanying paper is publicly available at `https://arxiv.org/abs/2504.10829` under the Creative Commons Attribution 4.0 International (CC BY 4.0) License.
- **CANDO code**: The implementation developed as part of this work is released at `https://github.com/Shell-OSS/cando` under the **Apache-2.0** License.
- **ALPS-Bench**: The benchmark introduced in this work is available at `https://github.com/Shell-OSS/cando/tree/main/data` under the **Apache-2.0** License. Any applicable access or usage restrictions are described in the accompanying dataset documentation.

NeurIPS Paper Checklist

1. **Claims**

   Question: Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope?

   Answer: [Yes]

   Justification: The three main claims stated in the introduction (**ALPS-Bench**, `CANDO`, and public-benchmark results) are supported by Sections 4, 5, and 6, Tables 1, 2, and 3, and the extended results in Appendices E.4 and E.5.

   Guidelines:

   - The answer [N/A] means that the abstract and introduction do not include the claims made in the paper.
   - The abstract and/or introduction should clearly state the claims made, including the contributions made in the paper and important assumptions and limitations. A [No] or [N/A] answer to this question will not be perceived well by the reviewers.
   - The claims made should match theoretical and experimental results, and reflect how much the results can be expected to generalize to other settings.
   - It is fine to include aspirational goals as motivation as long as it is clear that these goals are not attained by the paper.

2. **Limitations**

   Question: Does the paper discuss the limitations of the work performed by the authors?

   Answer: [Yes]

   Justification: Limitations are discussed in Section 7 and further elaborated in Appendix B.

   Guidelines:

   - The answer [N/A] means that the paper has no limitation while the answer [No] means that the paper has limitations, but those are not discussed in the paper.
   - The authors are encouraged to create a separate "Limitations" section in their paper.
   - The paper should point out any strong assumptions and how robust the results are to violations of these assumptions (e.g., independence assumptions, noiseless settings, model well-specification, asymptotic approximations only holding locally). The authors should reflect on how these assumptions might be violated in practice and what the implications would be.
   - The authors should reflect on the scope of the claims made, e.g., if the approach was only tested on a few datasets or with a few runs. In general, empirical results often depend on implicit assumptions, which should be articulated.
   - The authors should reflect on the factors that influence the performance of the approach. For example, a facial recognition algorithm may perform poorly when image resolution is low or images are taken in low lighting. Or a speech-to-text system might not be used reliably to provide closed captions for online lectures because it fails to handle technical jargon.
   - The authors should discuss the computational efficiency of the proposed algorithms and how they scale with dataset size.
   - If applicable, the authors should discuss possible limitations of their approach to address problems of privacy and fairness.
   - While the authors might fear that complete honesty about limitations might be used by reviewers as grounds for rejection, a worse outcome might be that reviewers discover limitations that aren't acknowledged in the paper. The authors should use their best judgment and recognize that individual actions in favor of transparency play an important role in developing norms that preserve the integrity of the community. Reviewers will be specifically instructed to not penalize honesty concerning limitations.

3. **Theory assumptions and proofs**

   Question: For each theoretical result, does the paper provide the full set of assumptions and a complete (and correct) proof?

Answer: [N/A]

Justification: The paper does not propose/present any formal theorems or lemmas requiring proofs. Definitions, Remarks and the objective in Eq. 2 are descriptive.

Guidelines:

- The answer [N/A] means that the paper does not include theoretical results.
- All the theorems, formulas, and proofs in the paper should be numbered and cross-referenced.
- All assumptions should be clearly stated or referenced in the statement of any theorems.
- The proofs can either appear in the main paper or the supplemental material, but if they appear in the supplemental material, the authors are encouraged to provide a short proof sketch to provide intuition.
- Inversely, any informal proof provided in the core of the paper should be complemented by formal proofs provided in appendix or supplemental material.
- Theorems and Lemmas that the proof relies upon should be properly referenced.

4. **Experimental result reproducibility**

Question: Does the paper fully disclose all the information needed to reproduce the main experimental results of the paper to the extent that it affects the main claims and/or conclusions of the paper (regardless of whether the code and data are provided or not)?

Answer: [Yes]

Justification: Algorithm 1 fully specifies the optimization loop. Agent roles, tools, and ICL initialization are described in Section 5 and Appendix F. Hyperparameters (N, B, K, W, and T) are reported in the ablation (Table 15), as well as in the implementation details of Section 6.

Guidelines:

- The answer [N/A] means that the paper does not include experiments.
- If the paper includes experiments, a [No] answer to this question will not be perceived well by the reviewers: Making the paper reproducible is important, regardless of whether the code and data are provided or not.
- If the contribution is a dataset and/or model, the authors should describe the steps taken to make their results reproducible or verifiable.
- Depending on the contribution, reproducibility can be accomplished in various ways. For example, if the contribution is a novel architecture, describing the architecture fully might suffice, or if the contribution is a specific model and empirical evaluation, it may be necessary to either make it possible for others to replicate the model with the same dataset, or provide access to the model. In general. releasing code and data is often one good way to accomplish this, but reproducibility can also be provided via detailed instructions for how to replicate the results, access to a hosted model (e.g., in the case of a large language model), releasing of a model checkpoint, or other means that are appropriate to the research performed.
- While NeurIPS does not require releasing code, the conference does require all submissions to provide some reasonable avenue for reproducibility, which may depend on the nature of the contribution. For example
  (a) If the contribution is primarily a new algorithm, the paper should make it clear how to reproduce that algorithm.
  (b) If the contribution is primarily a new model architecture, the paper should describe the architecture clearly and fully.
  (c) If the contribution is a new model (e.g., a large language model), then there should either be a way to access this model for reproducing the results or a way to reproduce the model (e.g., with an open-source dataset or instructions for how to construct the dataset).
  (d) We recognize that reproducibility may be tricky in some cases, in which case authors are welcome to describe the particular way they provide for reproducibility. In the case of closed-source models, it may be that access to the model is limited in some way (e.g., to registered users), but it should be possible for other researchers to have some path to reproducing or verifying the results.

5. **Open access to data and code**

   Question: Does the paper provide open access to the data and code, with sufficient instructions to faithfully reproduce the main experimental results, as described in supplemental material?

   Answer: [Yes]

   Justification: Code, prompt templates, evaluation assets, and the **ALPS-Bench** development set (900 layouts) are provided through the project page at `https://shell-oss.github.io/cando/`. The test-set ground truth is intentionally withheld to support a hosted online evaluation server.

   Guidelines:

   - The answer [N/A] means that paper does not include experiments requiring code.
   - Please see the NeurIPS code and data submission guidelines (`https://neurips.cc/public/guides/CodeSubmissionPolicy`) for more details.
   - While we encourage the release of code and data, we understand that this might not be possible, so [No] is an acceptable answer. Papers cannot be rejected simply for not including code, unless this is central to the contribution (e.g., for a new open-source benchmark).
   - The instructions should contain the exact command and environment needed to run to reproduce the results. See the NeurIPS code and data submission guidelines (`https://neurips.cc/public/guides/CodeSubmissionPolicy`) for more details.
   - The authors should provide instructions on data access and preparation, including how to access the raw data, preprocessed data, intermediate data, and generated data, etc.
   - The authors should provide scripts to reproduce all experimental results for the new proposed method and baselines. If only a subset of experiments are reproducible, they should state which ones are omitted from the script and why.
   - At submission time, to preserve anonymity, the authors should release anonymized versions (if applicable).
   - Providing as much information as possible in supplemental material (appended to the paper) is recommended, but including URLs to data and code is permitted.

6. **Experimental setting/details**

   Question: Does the paper specify all the training and test details (e.g., data splits, hyperparameters, how they were chosen, type of optimizer) necessary to understand the results?

   Answer: [Yes]

   Justification: The backbone models and GPU hardware are reported in Section 6; training and configuration details are provided in Appendix E. Evaluation metrics for our proposed benchmark are defined precisely in Section 4. Public-benchmark task definitions follow prior work.

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.
   - The experimental setting should be presented in the core of the paper to a level of detail that is necessary to appreciate the results and make sense of them.
   - The full details can be provided either with the code, in appendix, or as supplemental material.

7. **Experiment statistical significance**

   Question: Does the paper report error bars suitably and correctly defined or other appropriate information about the statistical significance of the experiments?

   Answer: [Yes]

   Justification: Table 1 reports means and standard deviations over the 100-layout **ALPS-Bench** test set. Appendix E.8 additionally reports paired two-sided Wilcoxon tests with Holm correction and paired-bootstrap 95% confidence intervals for AGG, CS, and RE.

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.

- The authors should answer [Yes] if the results are accompanied by error bars, confidence intervals, or statistical significance tests, at least for the experiments that support the main claims of the paper.
- The factors of variability that the error bars are capturing should be clearly stated (for example, train/test split, initialization, random drawing of some parameter, or overall run with given experimental conditions).
- The method for calculating the error bars should be explained (closed form formula, call to a library function, bootstrap, etc.)
- The assumptions made should be given (e.g., Normally distributed errors).
- It should be clear whether the error bar is the standard deviation or the standard error of the mean.
- It is OK to report 1-sigma error bars, but one should state it. The authors should preferably report a 2-sigma error bar than state that they have a 96% CI, if the hypothesis of Normality of errors is not verified.
- For asymmetric distributions, the authors should be careful not to show in tables or figures symmetric error bars that would yield results that are out of range (e.g., negative error rates).
- If error bars are reported in tables or plots, the authors should explain in the text how they were calculated and reference the corresponding figures or tables in the text.

8. **Experiments compute resources**

   Question: For each experiment, does the paper provide sufficient information on the computer resources (type of compute workers, memory, time of execution) needed to reproduce the experiments?

   Answer: [Yes]

   Justification: Hardware is summarized in Section 6; per-trial latency, parallel batch wall time, and local GPU-hours are reported in Appendix E.3.

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.
   - The paper should indicate the type of compute workers CPU or GPU, internal cluster, or cloud provider, including relevant memory and storage.
   - The paper should provide the amount of compute required for each of the individual experimental runs as well as estimate the total compute.
   - The paper should disclose whether the full research project required more compute than the experiments reported in the paper (e.g., preliminary or failed experiments that didn't make it into the paper).

9. **Code of ethics**

   Question: Does the research conducted in the paper conform, in every respect, with the NeurIPS Code of Ethics `https://neurips.cc/public/EthicsGuidelines`?

   Answer: [Yes]

   Justification: The research concerns automated layout design for operational facilities and raises no ethical concerns. The data does not include any personal identifiable information. The NeurIPS Code of Ethics has been reviewed.

   Guidelines:

   - The answer [N/A] means that the authors have not reviewed the NeurIPS Code of Ethics.
   - If the authors answer [No], they should explain the special circumstances that require a deviation from the Code of Ethics.
   - The authors should make sure to preserve anonymity (e.g., if there is a special consideration due to laws or regulations in their jurisdiction).

10. **Broader impacts**

    Question: Does the paper discuss both potential positive societal impacts and negative societal impacts of the work performed?

Answer: [Yes]

Justification: Potential positive and negative societal impacts are discussed in Appendix A.

Guidelines:

- The answer [N/A] means that there is no societal impact of the work performed.
- If the authors answer [N/A] or [No], they should explain why their work has no societal impact or why the paper does not address societal impact.
- Examples of negative societal impacts include potential malicious or unintended uses (e.g., disinformation, generating fake profiles, surveillance), fairness considerations (e.g., deployment of technologies that could make decisions that unfairly impact specific groups), privacy considerations, and security considerations.
- The conference expects that many papers will be foundational research and not tied to particular applications, let alone deployments. However, if there is a direct path to any negative applications, the authors should point it out. For example, it is legitimate to point out that an improvement in the quality of generative models could be used to generate Deepfakes for disinformation. On the other hand, it is not needed to point out that a generic algorithm for optimizing neural networks could enable people to train models that generate Deepfakes faster.
- The authors should consider possible harms that could arise when the technology is being used as intended and functioning correctly, harms that could arise when the technology is being used as intended but gives incorrect results, and harms following from (intentional or unintentional) misuse of the technology.
- If there are negative societal impacts, the authors could also discuss possible mitigation strategies (e.g., gated release of models, providing defenses in addition to attacks, mechanisms for monitoring misuse, mechanisms to monitor how a system learns from feedback over time, improving the efficiency and accessibility of ML).

11. **Safeguards**

Question: Does the paper describe safeguards that have been put in place for responsible release of data or models that have a high risk for misuse (e.g., pre-trained language models, image generators, or scraped datasets)?

Answer: [N/A]

Justification: The paper does not release a generative model with significant risk of misuse. `CANDO` is a task-specific layout design optimization framework and **ALPS-Bench** contains geometric facility layouts without personally identifiable or sensitive information.

Guidelines:

- The answer [N/A] means that the paper poses no such risks.
- Released models that have a high risk for misuse or dual-use should be released with necessary safeguards to allow for controlled use of the model, for example by requiring that users adhere to usage guidelines or restrictions to access the model or implementing safety filters.
- Datasets that have been scraped from the Internet could pose safety risks. The authors should describe how they avoided releasing unsafe images.
- We recognize that providing effective safeguards is challenging, and many papers do not require this, but we encourage authors to take this into account and make a best faith effort.

12. **Licenses for existing assets**

Question: Are the creators or original owners of assets (e.g., code, data, models), used in the paper, properly credited and are the license and terms of use explicitly mentioned and properly respected?

Answer: [Yes]

Justification: All existing assets (PubLayNet, RICO, PKU-PosterLayout, Qwen3.6, Qwen3-VL-8B-Instruct, GPT-5.2, LayoutDM, and LayoutCoT) are properly cited, and their licenses and terms of use are listed in Appendix G.

Guidelines:

- The answer [N/A] means that the paper does not use existing assets.
- The authors should cite the original paper that produced the code package or dataset.
- The authors should state which version of the asset is used and, if possible, include a URL.
- The name of the license (e.g., CC-BY 4.0) should be included for each asset.
- For scraped data from a particular source (e.g., website), the copyright and terms of service of that source should be provided.
- If assets are released, the license, copyright information, and terms of use in the package should be provided. For popular datasets, `paperswithcode.com/datasets` has curated licenses for some datasets. Their licensing guide can help determine the license of a dataset.
- For existing datasets that are re-packaged, both the original license and the license of the derived asset (if it has changed) should be provided.
- If this information is not available online, the authors are encouraged to reach out to the asset's creators.

13. **New assets**

    Question: Are new assets introduced in the paper well documented and is the documentation provided alongside the assets?

    Answer: [Yes]

    Justification: **ALPS-Bench** ($1,000$ annotated layouts released alongside a design manual) is a new asset introduced in the paper. Its annotation format and structure are described in Appendices D.2 and D.3.

    Guidelines:

    - The answer [N/A] means that the paper does not release new assets.
    - Researchers should communicate the details of the dataset/code/model as part of their submissions via structured templates. This includes details about training, license, limitations, etc.
    - The paper should discuss whether and how consent was obtained from people whose asset is used.
    - At submission time, remember to anonymize your assets (if applicable). You can either create an anonymized URL or include an anonymized zip file.

14. **Crowdsourcing and research with human subjects**

    Question: For crowdsourcing experiments and research with human subjects, does the paper include the full text of instructions given to participants and screenshots, if applicable, as well as details about compensation (if any)?

    Answer: [N/A]

    Justification: Annotation was performed by professional annotators working from original CAD drawings, not via a crowdsourcing platform. No human subjects research was conducted.

    Guidelines:

    - The answer [N/A] means that the paper does not involve crowdsourcing nor research with human subjects.
    - Including this information in the supplemental material is fine, but if the main contribution of the paper involves human subjects, then as much detail as possible should be included in the main paper.
    - According to the NeurIPS Code of Ethics, workers involved in data collection, curation, or other labor should be paid at least the minimum wage in the country of the data collector.

15. **Institutional review board (IRB) approvals or equivalent for research with human subjects**

Question: Does the paper describe potential risks incurred by study participants, whether such risks were disclosed to the subjects, and whether Institutional Review Board (IRB) approvals (or an equivalent approval/review based on the requirements of your country or institution) were obtained?

Answer: [N/A]

Justification: The paper does not involve research with human subjects and does not require IRB approval.

Guidelines:

- The answer [N/A] means that the paper does not involve crowdsourcing nor research with human subjects.
- Depending on the country in which research is conducted, IRB approval (or equivalent) may be required for any human subjects research. If you obtained IRB approval, you should clearly state this in the paper.
- We recognize that the procedures for this may vary significantly between institutions and locations, and we expect authors to adhere to the NeurIPS Code of Ethics and the guidelines for their institution.
- For initial submissions, do not include any information that would break anonymity (if applicable), such as the institution conducting the review.

16. **Declaration of LLM usage**

Question: Does the paper describe the usage of LLMs if it is an important, original, or non-standard component of the core methods in this research? Note that if the LLM is used only for writing, editing, or formatting purposes and does *not* impact the core methodology, scientific rigor, or originality of the research, declaration is not required.

Answer: [Yes]

Justification: LLMs are a central and original component of the `CANDO` framework. Qwen and GPT serve as backbone models (Section 6) for all of the proposed solution's agents. Their roles, versions, and usage are described throughout Section 5.

Guidelines:

- The answer [N/A] means that the core method development in this research does not involve LLMs as any important, original, or non-standard components.
- Please refer to our LLM policy in the NeurIPS handbook for what should or should not be described.