FROM REQUIREMENTS AND DRAWINGS TO OPTIMIZED WAREHOUSE FLOORPLANS: LLM-DRIVEN FORMULATION AND SOLVER-BASED DESIGN

Anonymous authors
Paper under double-blind review

ABSTRACT

Warehouse layout design must translate natural-language requirements into feasible spatial decisions while accounting for operational performance. We present FLOWS (Formulation-Led Optimization of Warehouses with Solvers), a neuro-symbolic framework that connects language-based modeling with symbolic optimization through an explicit integer-programming formulation. A formulation agent grounds decision variables, objectives, and constraints in a graph representation of warehouse geometry and a finite rack catalog. A separate strategy agent selects admissible solution procedures without changing the underlying problem. Candidate layouts undergo static validation and simulation-based acceptance; diagnostic feedback supports formulation revision while preserving the original requirements. This separation makes the model, solution procedure, and acceptance decision independently inspectable. With replayed formulations and a fixed constructive portfolio, FLOWS produces verified layouts on all 42 warehouse plans and improves mean operating performance over two adapted planners. Small component ablations identify declaration-repair gains but do not establish an advantage for adaptive strategy selection or operational feedback.

1 INTRODUCTION

As a critical link among production, distribution, and order fulfillment, warehouses play an essential role in the modern logistics industry. Industry forecasts and building-retrofit benchmarks suggest that more than 10,000 large warehouse facilities worldwide may be newly constructed or renovated each year (Interact Analysis, 2022). Warehouses depend on spatial organization to provide storage capacity, reduce material-handling effort, and respond to delivery demands (Rouwenhorst et al., 2000; Gu et al., 2007; de Koster et al., 2007). These objectives compete for limited space: allocating more area to storage can restrict access and circulation, whereas wider aisles and more direct routes leave less room for racks. Changing storage, fulfillment, and automation requirements further demand that layouts evolve, often within fixed irregular building envelopes and around existing infrastructure (Baker & Canessa, 2009; Boysen et al., 2019; Azadeh et al., 2019). Warehouse construction and reconfiguration therefore require balancing density, accessibility, and operating performance under facility-specific constraints.

The central difficulty is the coupling between requirement formalization, spatial configuration, and operational evaluation. Natural-language requirements must become precise constraints; rack and aisle arrangements must satisfy geometric and connectivity conditions; and feasible layouts must be assessed for their operational consequences. These decisions define a combinatorial search space in which local changes have nonlocal effects: adding a rack may improve capacity while blocking a route, and restoring connectivity may alter traffic patterns and handling effort. Design therefore involves repeated cycles of formulation, search, and validation. Existing approaches address parts of this process through analytical design, mathematical optimization, simulation, and constructive aisle generation (Baker & Canessa, 2009; Heragu et al., 2005; Gu et al., 2010; Gue & Meller, 2009; Öztürkoğlu et al., 2014). However, prescribed layout families require adaptation to each facility, and validation can expose access or connectivity failures without determining how to revise the search space (Shahroudnejad et al., 2024). Coordinating these tools and propagating feedback into subsequent decisions remains a substantial planning burden.

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

Large language models offer a potential coordination mechanism. Their ability to interpret requirements can support executable optimization modeling (Ahmed & Choudhury, 2024), while solver orchestration and feedback-guided revision can connect formulation with solution search (Ahmaditeshnizi et al., 2024). For warehouse planning, this suggests assigning language models three responsibilities: formalizing facility-specific requirements, selecting suitable solution procedures, and interpreting validation evidence to propose revisions. Explicit interfaces constrain these decisions, deterministic checks establish mathematical and physical validity, and simulation evaluates operational performance. The resulting architecture can accommodate varied design requests while keeping feasibility and acceptance independently checkable.

Motivated by the need to coordinate language-based planning with independently checkable feasibility and acceptance, we introduce FLOWS (Formulation-Led Optimization of Warehouses with Solvers). A formulation agent generates an integer program over finite, catalog-grounded rack placements on a warehouse access graph. A separate strategy agent selects an admissible solution procedure without changing the problem, and layered validation guides bounded revisions while preserving the original requirements.

Our contributions are threefold: an executable formulation interface grounding language-generated decisions in deterministic geometry; a planning architecture separating formulation, solver control, and acceptance; and an evaluation linking layout availability, storage utilization, operational performance, and computational cost. Experiments demonstrate that, with replayed formulations and a fixed constructive portfolio, FLOWS produces physically verified layouts on all 42 floorplans at 32.28% mean utilization, with mean throughput gains of 4.13% over CBS-24 and 9.93% over CPU-PES-25, and corresponding mean cycle-time reductions of 4.96% and 10.77%. Separate component ablations examine strategy selection and feedback. We next formalize the warehouse representation and acceptance conditions that support these interfaces.

2 PRELIMINARIES

2.1 WAREHOUSE DESIGN AND NEURO-SYMBOLIC OPTIMIZATION

Warehouse design couples physical layout, storage capacity, and material-handling performance (Rouwenhorst et al., 2000; Gu et al., 2010). Analytical and constructive approaches exploit prescribed aisle families, while graph-based methods make access and connectivity explicit (Gue & Meller, 2009; Öztürkoğlu et al., 2014; Shahroudnejad et al., 2024). Discrete-event simulation evaluates the resulting layouts under stochastic demand and finite handling resources (Azadeh et al., 2019; Amaran et al., 2016). These components address different aspects of the task: a connected layout need not meet an operational target, and an operationally attractive layout may violate a physical requirement.

Language-to-optimization systems generate mathematical models for execution by optimization software (Ahmaditeshnizi et al., 2024; Huang et al., 2025). Related neuro-symbolic systems separate language interpretation from logical inference (Pan et al., 2023; Olausson et al., 2023). We adopt this division of responsibility for warehouse planning, with an explicit distinction between the problem formulation, the procedure used to solve it, and the evidence required to accept a layout. A well-formed formulation is executable; its fidelity to the user's requirements remains a separate property. In particular, a solver certificate is conditional on the translated model and cannot establish that every requested requirement was translated correctly. This motivates retaining requirement provenance and fixing the acceptance conditions independently of the agent's choice of solver.

2.2 FACILITY GRAPH AND LAYOUT DECISIONS

An instance consists of annotated facility geometry Ω derived from a drawing, a rack catalog K, natural-language requirements R, and an operating protocol W. The geometry specifies storage and navigation regions, obstacles, and fixed terminals such as loading docks. The catalog fixes rack dimensions, admissible orientations, service faces, and storage capacities. The protocol specifies demand and resource assumptions for evaluating operational requirements; drawing interpretation precedes the formulation process studied here. Missing assumptions remain unresolved rather than being supplied by the optimization objective.

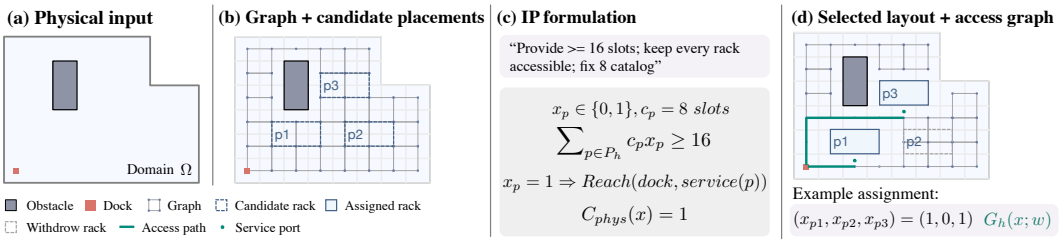

Figure 1: Illustrative mapping from geometry to a discrete warehouse representation. (a) Fixed facility geometry with an 8×6 m bounding box and 2×1 m catalog racks, each providing $c_p = 8$ slots. (b) A four-neighbor graph at $h = 1$ m; the displayed $P = \{p_1, p_2, p_3\}$ is a subset of P_h. (c) Binary placement decisions subject to capacity, access, and physical constraints; Reach is evaluated in the layout-dependent graph. (d) The illustrative assignment $(1, 0, 1)$ provides 16 slots and changes the access graph at clearance width $w = 0.6$ m. Short graph-to-service connectors are omitted. The example illustrates representation and feasibility, without evaluating optimality or operational KPIs.

A geometry kernel discretizes the navigation region into a four-neighbor graph $G_h = (V_h, E_h)$ at resolution h. Each retained cell contributes a node, and adjacent cells may be connected horizontally or vertically. A finite placement set P_h associates catalog racks with admissible grid anchors. The decision $x_p \in \{0, 1\}$ indicates whether placement $p \in P_h$ is selected; its physical footprint and capacity c_p are fixed input facts. Thus a layout is represented by $x \in \{0, 1\}^{|P_h|}$, rather than by coordinates generated by the language model.

Selected racks alter traversability. For a required clearance w, write $G_h(x; w)$ for the graph after accounting for rack footprints, obstacles, and clearance, including connections to terminals and rack service faces. Access is a property of this layout-dependent graph, not merely of the initial free-space grid. Storage and navigation regions can differ, allowing a main aisle to remain traversable while being excluded from rack placement. All statements about feasibility and optimality are conditional on the chosen finite representation; finer grids do not automatically preserve every placement or connection. Figure 1 distinguishes navigation nodes from placement decisions: selecting p_1 and p_3 supplies 16 slots, but the capacity bound alone does not force p_2 to be unselected.

2.3 Static formulation and operational acceptance

Let $\mathcal{C}_{\text{phys}}(x; \Omega, K)$ denote mandatory physical conditions, and let $\mathcal{C}_R(x)$ denote the intended static requirements expressed by R. Let $J_R(x)$ be the stated design objective, with maximization used below for notation, and let $k_j(x; W)$ denote an operational quantity under the fixed protocol. The complete design task is

$$\begin{aligned} \max_{x \in \{0,1\}^{|P_h|}} \quad & J_R(x) \\ \text{s.t.} \quad & \mathcal{C}_{\text{phys}}(x; \Omega, K) = 1, \quad \mathcal{C}_R(x) = 1, \\ & k_j(x; W) \ \bowtie_j \ \tau_j, \qquad j = 1, \ldots, m, \end{aligned} \tag{1}$$

where $\bowtie_j$ and τ_j specify the direction and threshold of each service requirement. A request may instead prescribe minimization or ask only for a feasible layout; the system preserves that choice.

Binary placement decisions and supported finite constraints admit an integer-programming (IP) formulation of the *static* problem, with physical relations supplied by the graph representation. For a generated formulation F, deterministic compilation yields a bounded model of the form

$$\mathcal{M}(F): \quad \max_{x,z} \ f_F(x, z) \quad \text{s.t.} \quad A_F x + B_F z \leq b_F, \quad x \in \{0,1\}^{|P_h|}, \quad z \in \mathcal{Z}_F, \tag{2}$$

where $\mathcal{Z}_F$ is a finite integer domain for auxiliary variables and f_F is the compiled linear objective. Supported logical and graph constraints are integer-encodable relations. The operational conditions in Equation 1 are evaluated through simulation and are not asserted to have an exact linear encoding. Consequently, faithful formulation, static feasibility, static optimality, and operational acceptance are distinct outcomes. The method connects these outcomes through explicit formulation, solver, and validation interfaces; Appendix A specifies their concrete encodings.

3 METHOD

3.1 OVERVIEW: FORMULATION, SOLUTION, AND VALIDATION

FLOWS organizes warehouse planning around three coupled decisions: which integer program represents the request, how that program should be solved, and whether its solution meets the original design requirements. A *formulation agent* generates the IP formulation from the requirements and geometry-bound facts. A separate *strategy agent* selects an admissible solution procedure for that formulation. A deterministic validation layer checks the resulting layout and evaluates its operational performance, returning diagnostic evidence when another iteration is warranted. The two agents have distinct roles and contexts and may use separate calls to the same pretrained model. Figure 2 summarizes the three stages and their feedback paths.

Let $\mathcal{I}_h = (\Omega, K, G_h, P_h, W)$ collect the fixed geometry, catalog, discrete representation, and operating protocol. At iteration t, the forward computation is

$$(R, \mathcal{I}_h) \longrightarrow F_t \longrightarrow (F_t, S_t) \longrightarrow (x_t, \sigma_t) \longrightarrow V_t, \tag{3}$$

where S_t is the solver plan, σ_t records the solver's status and any certified bound, and V_t contains validation evidence. The formulation defines the optimization problem, the plan defines an execution policy, and the validator determines acceptance. This separation permits changes to a solution procedure without silently changing the problem, and changes to search hypotheses without silently changing the acceptance criteria.

3.2 STAGE I: GENERATING THE INTEGER-PROGRAMMING FORMULATION

The formulation agent maps the requirements and instance data to an initial formulation, $F_0 = \pi_{\text{form}}(R, \mathcal{I}_h)$; later iterations additionally use validation feedback. Its output is an *integer-programming formulation*: named index sets, bound coefficients, integer decision domains, an objective, and static constraints, with operational targets retained as associated evaluation obligations. Each requirement-bearing declaration records its supporting input requirement or assumption. The typed internal representation makes this mathematical object executable and preserves the distinction between authored requirements and geometry-bound facts.

The static problem specified by F_t has the general form

$$\begin{aligned} \max_{x \in \{0,1\}^{|P_h|}} \quad & J_t(x) = \sum_{\ell=1}^{q_t} \lambda_{t\ell}\, \psi_{t\ell}(x) \\ \text{s.t.} \quad & \mathcal{C}_{\text{phys}}(x; \Omega, K) = 1, \\ & \mathcal{C}_{t,i}(x; G_h, P_h) = 1, \qquad i \in \mathcal{Q}_t. \end{aligned} \tag{4}$$

Here q_t counts the supported objective terms $\psi_{t\ell}$, and $\lambda_{t\ell}$ encode declared priorities and unit-consistent scaling; minimization uses a sign change and feasibility uses a constant objective. The index set $\mathcal{Q}_t$ collects the authored static requirements. Each predicate $\mathcal{C}_{t,i}$ can express an indexed inequality, a spatial relation, or a supported graph condition, such as reachability in $G_h(x; w)$. For example, $\psi(x) = \sum_{p \in P_h} c_p x_p$ measures capacity, while $\sum_{p \in P_r} c_p x_p \geq D_r$ expresses a regional capacity floor: P_r contains placements satisfying the membership rule for region r, and D_r is its required capacity. Objective terms and constraints must belong to the finite modeling language; simulation KPIs remain evaluation obligations rather than assumed linear terms. This distinction preserves the strength of a request: a minimum capacity belongs in the feasible-set definition, whereas a preference for greater capacity belongs in the objective. Replacing the former by an objective reward would allow other preferences to compensate for violating a mandatory requirement.

The compiler binds the declared sets and coefficients to deterministic geometry, checks units and references, and produces $\mathcal{M}(F_t) = \text{Compile}(F_t; \mathcal{I}_h)$, the finite IP of Equation 2. This lowering preserves $J_t(x)$ through the compiled objective $f_{F_t}(x, z)$; auxiliary variables z encode logical and graph relations without adding new layout decisions. Physical containment, non-overlap, and preservation of fixed terminals are enforced independently of the generated requirement list. The agent cannot change catalog dimensions or provide replacement coordinates. Ambiguous or unsupported requirements are reported before solving; executability alone does not establish semantic fidelity. Once compiled, the formulation becomes the fixed input to strategy selection.

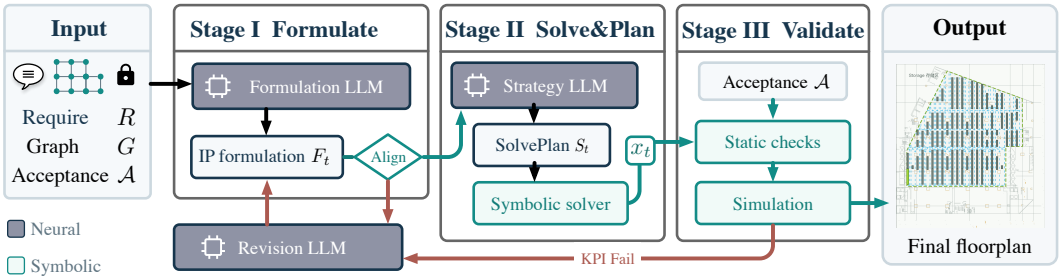

Figure 2: Three-stage architecture of FLOWS. A formulation LLM produces F_t from requirements and graph-grounded facts. After alignment with fixed acceptance conditions $\mathcal{A}$, a strategy LLM selects SolvePlan S_t and a symbolic solver produces x_t. Static checks and simulation supply revision evidence; output requires independent final evaluation. Budget handling and terminal statuses are omitted. Output panels illustrate layouts, not measured KPIs.

3.3 Stage II: Selecting a Solution Strategy

The strategy agent conditions its decision on the formulation, a deterministic structural profile, and an external resource budget:

$$S_t = \pi_{\text{solve}}\big(F_t, \phi(F_t, G_h), B_t\big). \tag{5}$$

The profile describes the model's size and supported constraint structure. The resulting plan selects from implemented procedures and is bound to the unchanged formulation. It cannot redefine the objective, remove a requirement, or authorize an arbitrary generated program. The purpose of this stage is to adapt computational effort to model structure; whether that adaptation is faster is an empirical question that includes the cost of obtaining the plan.

Given S_t, the symbolic solver executes the selected procedure on the compiled model:

$$(x_t, \sigma_t) = \text{Solve}\big(\mathcal{M}(F_t); S_t, B_t\big), \qquad x_t \in \mathcal{X}(F_t) \cup \{\perp\}, \tag{6}$$

where $\mathcal{X}(F_t)$ is the projection of the compiled feasible set onto the placement variables. The solver decodes a checked integer assignment into x_t, discarding encoding auxiliaries; $\perp$ denotes that no checked candidate is available. The record σ_t distinguishes a feasible incumbent from a proven optimum and retains any certified bound. Thus the plan controls how the model is solved, while the formulation determines which layouts may be returned.

Our implementation provides a complete connectivity encoding and an alternative that enforces supported positive connectivity requirements incrementally. The latter first solves a relaxation, checks the proposed layout, and adds valid connectivity constraints when a selected rack is disconnected. An authorized fallback returns to the complete encoding when incremental enforcement is unsuitable or exhausts its limits. Appendix A.3 derives the connectivity cuts and their applicability conditions.

Strategy changes preserve result semantics. Equivalent encodings retain feasible layouts and objective values; relaxations provide certified bounds but require checking all omitted requirements before returning a candidate. Optimality concerns only the static model, and budget exhaustion does not establish infeasibility. The LLM selects the procedure; deterministic components construct constraints and verify results before the candidate enters acceptance testing. For positive access requirements, the strategic tradeoff is whether to represent connectivity in advance or enforce it as violations are discovered. The former pays the encoding cost before search; the latter may use a smaller initial model but require repeated solves. Structural profiling informs this choice but does not guarantee that either procedure will succeed within a particular budget.

3.4 Stage III: Acceptance and Feedback

Stage III produces validation evidence $V_t = \text{Validate}(x_t, \sigma_t; \mathcal{A})$ against fixed acceptance conditions $\mathcal{A}$: a reference formulation F^{acc}, operational targets $\mathcal{T}$, and protocol W bound to the original instance. The reference is independently reviewed when available; otherwise it is F_0, making acceptance conditional on the initial interpretation. An independent check requires $x_t \in \mathcal{X}_{\text{stat}}(\mathcal{A})$,

the set satisfying physical constraints and the reference formulation, before simulation. Feasibility for the current search formulation alone is insufficient; without operational targets, acceptance is static-only.

When operational targets are present, simulation estimates performance of a statically valid layout under the prescribed workload and resources. Acceptance requires both forms of evidence:

$$\mathrm{Accept}(x_t; \mathcal{A}) = \mathbb{I}[x_t \in \mathcal{X}_{\mathrm{stat}}(\mathcal{A})] \cdot \mathbb{I}[\mathrm{Sim}(x_t; W) \models \mathcal{T}]\,. \tag{7}$$

Here $\models$ denotes satisfaction under the evaluation protocol, including uncertainty assessment and independent final confirmation. The evidence V_t records the checked conditions, KPI estimates, and remaining violations or uncertainty. A search-time pass triggers final evaluation on separate random streams; inconclusive or failed final evidence does not yield an accepted layout and is never used for further adaptation.

For a correctable declaration mismatch or a definite search-time KPI failure, the formulation agent uses V_t to propose an admissible revision. The next iteration then reconnects the three stages:

$$F_{t+1} = \mathrm{Revise}(F_t, V_t; \mathcal{A}), \qquad (F_{t+1}, B_{t+1}) \xrightarrow{\text{Stage II}} (S_{t+1}, x_{t+1}, \sigma_{t+1}). \tag{8}$$

Revision restores reference-aligned declarations or changes reversible search restrictions while preserving $\mathcal{A}$; it cannot weaken the original requirements or redefine the workload. A simulation failure supports a search hypothesis, not a proof that an entire family of layouts is invalid. Any additional restriction therefore remains reversible, and its resulting candidate must again pass the same acceptance conditions. The cycle returns an accepted layout when all conditions pass, or a documented stopping outcome when evidence or resources are insufficient. Solver-budget failures may instead return to Stage II, while checker disagreements stop execution. Appendix A.4 specifies the bounded controller and statistical acceptance rules. The experiments next distinguish warehouse-scale performance of a fixed-portfolio instantiation from component-level evidence for strategy selection and revision.

4 Experiments

Setup and scope. We evaluate FLOWS on 42 annotated warehouse floorplans against adapted constrained beam search (CBS-24; Shahroudnejad et al., 2024), adapted zoning (CPU-PES-25; Looms & Xie, 2025), and archived direct LLM generation. The main experiment replays one LLM formulation per case and uses a fixed constructive portfolio with development-simulation selection under a 300-second local planning budget. Physically verified layouts receive 20 paired confirmation replications on streams separate from selection, with results averaged within and then across warehouses. The workload is a finite-horizon overload test on development-exposed cases; capacities and destination distributions are not matched. Appendices B–C specify the protocol; Appendix F lists every case, KPI, and runtime.

4.1 Storage and Operating Performance

With a uniform 0.33 construction target, FLOWS produces 42 physically verified layouts at 32.28% mean footprint utilization, compared with 31.11% for CBS and 15.23% for CPU (Table 1). Of these, 37/42 attain the construction target and 39/42 attain 32%; all below-target layouts remain in the comparison. Mean throughput improves by 4.13% over CBS and 9.93% over CPU, while mean cycle time decreases by 4.96% and 10.77%, respectively. The configuration thus combines higher mean storage utilization and operating performance, with case-level trade-offs examined below.

Runtime–quality trade-off. Figure 3 shows the additional cost of FLOWS's formulation and verified search. Local planning averages 168.44 s and finishes within 300 s on all 42 cases; including archived generation and final confirmation gives a 291.07 s recorded component sum. For offline warehouse design, this minutes-scale cost is a reasonable trade-off for the observed gains in storage utilization and operating performance.

Travel and vehicle service time. The portfolio varies rack-bank length and orientation while preserving aisle access; simulation selects feasible layouts by operating consequences as well as

Table 1: Main comparison on all 42 cases, with 20 paired replications per layout. Bold marks best values, not significance; FLOWS uses formulation replay and a fixed portfolio.

Method	Util. ↑ (ratio)	Throughput ↑ (orders/h)	Mean cycle ↓ (min)	P90 cycle ↓ (min)	Edge block ↓ (min)	Travel ↓ (m)
CBS-24	0.3111	66.04	322.62	578.10	0.120	175.95
CPU-PES-25	0.1523	62.55	343.64	616.24	0.145	191.65
FLOWS	**0.3228**	**68.77**	**306.63**	**548.96**	**0.091**	**147.82**

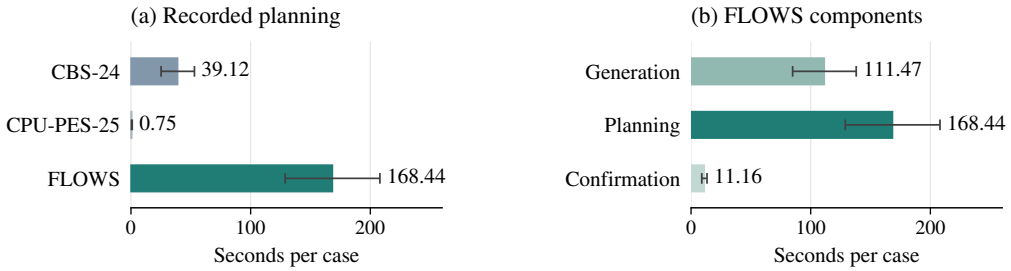

Figure 3: Runtime means with case-level SD bars. Planning covers 40 CBS cases and 42 CPU/FLOWS cases; FLOWS components each cover 42. Baseline planner calls and FLOWS's compilation, checking, and development simulation have different timing boundaries.

utilization. Mean travel falls by 15.99% relative to CBS and 22.87% relative to CPU; shorter travel and higher throughput coincide in 30/42 and 36/42 cases, respectively. With fleet utilization near 99.9%, shorter trips reduce vehicle service time and support higher completion capacity. Edge blocking contributes much less to the service-time differences, so the gains are consistent primarily with shorter travel; they do not establish a causal reduction in congestion.

Case-level storage–performance trade-offs. Against CPU, FLOWS has higher utilization on all 42 cases and higher throughput on 36. Against CBS, FLOWS improves throughput on 13 of the 21 cases with nonlower utilization, but this group's mean difference is -0.10 orders/h; the other 21 average $+5.55$ orders/h at lower utilization. The overall throughput-difference interval includes zero. These trade-offs motivate the casewise utilization floors below; group means alone do not establish density-preserving or causal component gains.

4.2 Feasibility and Direct LLM Generation

We next assess direct-generation availability. Direct generators output rack coordinates in one attempt without symbolic search or KPI feedback. FLOWS provides 42/42 available layouts versus 15/42 for each direct generator (Table 2). Availability requires timely output, valid rack geometry, all-rack access, and completed simulation, rather than operational-target satisfaction.

Table 2: Availability on all 42 inputs and mean runtime over available layouts only. Runtime sums recorded generation, planning (when used), and confirmation; cohorts and timing scopes differ.

Method	Available ↑	Rate ↑	Runtime (s)
DeepSeek	15/42	35.71%	140.18
GPT6	15/42	35.71%	252.61
FLOWS	42/42	**100.00%**	291.07

Geometry-grounded feasibility. DeepSeek's 27 rejected layouts comprise four dimension violations, 15 geometry failures, and eight reachability failures. These are relations FLOWS binds to catalog facts and checks during construction, supporting the use of geometry-grounded decisions and explicit access verification. GPT6's 27 failures are generation timeouts; all timely outputs pass physical checks. Different models and budget boundaries prevent equal-budget or isolated causal claims.

Among successful direct outputs, FLOWS also improves the observed storage–performance balance (Table 3). On DeepSeek's 15-case cohort, the main 0.33 configuration raises mean utilization from 0.3220 to 0.3304 and throughput by 12.26%. On GPT6's different 15-case cohort, a separate 0.34 configuration raises utilization from 0.3325 to 0.3382 and throughput by 3.14%. Mean cycle time decreases by 12.04% and 4.60%, respectively. These averages condition on each generator's successful outputs.

Table 3: Direct generation versus FLOWS on separate common-case cohorts ($n = 15$ per block), using 20 paired confirmation replications. Bold marks the best KPIs within each block. FLOWS subscripts denote construction targets (0.33/0.34). Tokens are historical per-case usage: one DeepSeek request, the full GPT6 subtask, or FLOWS's inherited formulation request; scopes differ.

Method	Util. ↑ (ratio)	Throughput ↑ (orders/h)	Mean cycle ↓ (min)	P90 cycle ↓ (min)	Edge block ↓ (min)	Travel ↓ (m)	Tokens (k/case)
(a) DeepSeek–FLOWS common cases ($n = 15$)							
DEEPSEEK	0.3220	59.33	363.49	650.96	0.190	223.29	41.66
$\text{FLOWS}_{.33}$	**0.3304**	**66.61**	**319.71**	**572.44**	**0.089**	**166.41**	11.07
(b) GPT6–FLOWS common cases ($n = 15$; different cohort)							
GPT6	0.3325	70.63	295.45	528.63	0.111	131.99	98.20
$\text{FLOWS}_{.34}$	**0.3382**	**72.85**	**281.86**	**504.05**	**0.088**	**122.89**	12.05

Token counts describe different archived workflows and are not a controlled measure of model efficiency. Appendix C gives generation accounting and the shared-eight-case comparison.

4.3 Operating Performance Under Comparator Utilization Floors

To address the storage–performance trade-off directly, each comparator's exact utilization sets a casewise constraint $U(L^{\text{FLOWS}}) \geq U(L^{\text{base}})$. Only this scalar enters planning, not comparator coordinates or KPIs. Qualified pairs receive new confirmation replications independent of selection; the seven unchanged below-floor layouts receive diagnostic simulations but remain failures, excluded from qualified means. This controls footprint utilization; capacity and destination demand remain unmatched.

Results. Against CBS, CPU, DeepSeek and GPT6, FLOWS meets the utilization floor in 39/42, 42/42, 11/15, 15/15 tasks, respectively. All floor-qualified layouts complete confirmation, giving 107 successful pairs out of 114 tasks. On these pairs, mean throughput differences (FLOWS minus comparator) are $+5.15, +16.58, +8.29, +3.70$ orders/h (Figure 5). The seven below-floor layouts also appear in the figure using additional diagnostic simulations; they remain failures and are excluded from these conditional means. Appendix D reports their utilization shortfalls and throughput, together with uncertainty for the successful pairs.

On the floor-qualified pairs, positive mean gains persist without reducing storage footprint; mean utilization excess is 0.037–0.048 percentage points across groups. The seven failures limit this conclusion to 107 floor-qualified tasks; Figure 4 shows C07 and Appendix G provides further successful and failed layouts.

4.4 Component Ablations

The preceding comparisons evaluate complete layout workflows, with direct generation providing a joint control without explicit formulation or symbolic search. Two archived synthetic cohorts separately examine strategy selection and feedback (Appendix E).

On 12 IP instances, fixed exact solving preserves every optimal objective while taking 27.4 ms versus 86.0 ms for LLM-selected plans; including plan generation, the latter takes 37.57 s (case-level medians). On the separate 12-case repair suite, removing feedback reduces acceptance from 9/12 to 4/12 with validation retained. Deterministic reference restoration also achieves 9/12: all gains concern declaration repair, and no operational case triggers KPI revision. These tests validate solver execution and declaration repair, but establish neither acceleration nor an LLM-feedback advantage over reference restoration.

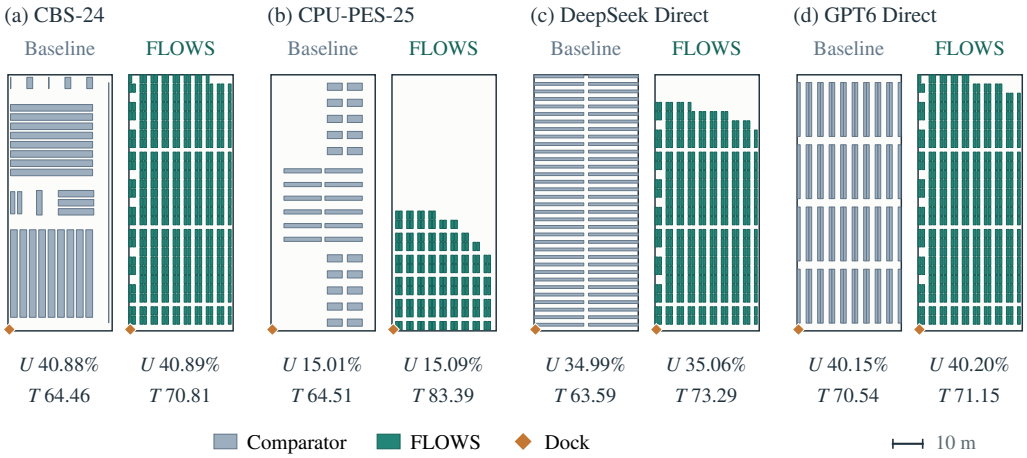

Figure 4: C07, selected for higher throughput in all four qualified pairs. Comparators (left) and FLOWS layouts under their utilization floors (right) share one scale. Orange diamonds mark docks; U is utilization and T throughput (orders/h). In (b), FLOWS adds feasible placements nearest the dock until reaching the floor, concentrating storage nearby. Relative to CPU, travel falls from 154.33 to 69.93 m, throughput rises 29.3%, and mean cycle time falls 33.7%.

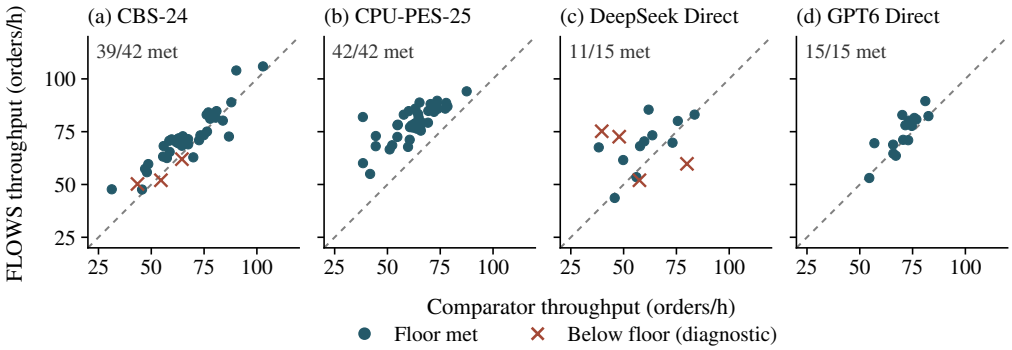

Figure 5: Throughput with each comparator's exact per-case utilization imposed as FLOWS's hard floor. All 114 eligible tasks are shown; each point averages 20 replications on paired streams. Teal circles mark the 107 floor-qualified successful pairs; red crosses show additional diagnostic evaluations of all seven below-floor layouts. Points above the diagonal have higher FLOWS throughput, but diagnostic crosses remain constraint failures.

5 Conclusion

FLOWS connects warehouse requirements and geometry to language-generated integer-programming formulations, symbolic solution procedures, and independent validation. Separating formulation, solver control, and acceptance makes their roles explicit while preserving the original requirements across revisions. This provides a common interface for combining language interpretation with established optimization and simulation tools. With replayed formulations and a fixed constructive portfolio, it produces verified layouts on all 42 cases at 32.28% mean utilization, with higher mean throughput and lower mean cycle time than two adapted planners. Comparator-specific utilization floors retain positive mean operating gains across the 107 qualified tasks. Component ablations find no strategy speedup and declaration-repair gains matched by deterministic restoration; operational-feedback benefits and generalization of the complete adaptive loop remain open.

AI USE STATEMENT

Generative AI tools assisted the conceptual framework and mathematical formulation, the problem-formulation interface, solver and validation software, experiment designs, result analyses, figure development, and manuscript text and LaTeX preparation. Language models are also evaluated components of the planning system and direct-generation comparisons; the experimental protocols distinguish their roles, archived-call provenance, and recorded usage from subsequent deterministic replay and simulation. The project records model outputs, compilation and validation results, source bindings, and simulation replications for the reported experiments. These checks support artifact consistency and implemented constraint satisfaction; they do not by themselves verify every language interpretation or scientific claim. The authors retain responsibility for the manuscript, its claims, and the associated artifacts.

ETHICS STATEMENT

The reported evaluations concern computational layout planning and simulation rather than a study involving human participants. Warehouse instances are identified by anonymized case labels in the manuscript. Facility layouts can contain sensitive operational information, so public release of source drawings requires an appropriate data-rights and privacy review. The implemented geometry, access, and simulation checks do not certify compliance with building codes, fire safety, equipment standards, or real-world collision avoidance; deployment requires qualified engineering review.

REPRODUCIBILITY STATEMENT

Section 4 summarizes the comparison cohorts, utilization targets, and evaluation design. Appendices A–E supply the mathematical encodings, execution and evaluation protocols, comparator provenance, and component studies; Appendix F lists complete case-level results. The local experiment records retain per-case configurations, layout and source hashes, raw simulation replications, and reporting scripts. The manuscript does not claim that these artifacts or the original facility drawings have already been publicly released.

REFERENCES

Ali Ahmaditeshnizi, Wenzhi Gao, and Madeleine Udell. OptiMUS: Scalable optimization modeling with (MI)LP solvers and large language models. In *Proceedings of the 41st International Conference on Machine Learning*, volume 235 of *Proceedings of Machine Learning Research*, pp. 577–596. PMLR, 2024. URL https://proceedings.mlr.press/v235/ahmaditeshnizi24a.html.

Tasnim Ahmed and Salimur Choudhury. LM4OPT: Unveiling the potential of large language models in formulating mathematical optimization problems. *INFOR: Information Systems and Operational Research*, 62(4):559–572, 2024. doi: 10.1080/03155986.2024.2388452.

Satyajith Amaran, Nikolaos V. Sahinidis, Bikram Sharda, and Scott J. Bury. Simulation optimization: A review of algorithms and applications. *Annals of Operations Research*, 240(1):351–380, 2016. doi: 10.1007/s10479-015-2019-x.

Kaveh Azadeh, René de Koster, and Debjit Roy. Robotized and automated warehouse systems: Review and recent developments. *Transportation Science*, 53(4):917–945, 2019. doi: 10.1287/trsc.2018.0873.

Peter Baker and Marco Canessa. Warehouse design: A structured approach. *European Journal of Operational Research*, 193(2):425–436, 2009. doi: 10.1016/j.ejor.2007.11.045.

Nils Boysen, René de Koster, and Felix Weidinger. Warehousing in the e-commerce era: A survey. *European Journal of Operational Research*, 277(2):396–411, 2019. doi: 10.1016/j.ejor.2018.08.023.

René de Koster, Tho Le-Duc, and Kees Jan Roodbergen. Design and control of warehouse order picking: A literature review. *European Journal of Operational Research*, 182(2):481–501, 2007. doi: 10.1016/j.ejor.2006.07.009.

Jinxiang Gu, Marc Goetschalckx, and Leon F. McGinnis. Research on warehouse operation: A comprehensive review. *European Journal of Operational Research*, 177(1):1–21, 2007. doi: 10.1016/j.ejor.2006.02.025.

Jinxiang Gu, Marc Goetschalckx, and Leon F. McGinnis. Research on warehouse design and performance evaluation: A comprehensive review. *European Journal of Operational Research*, 203(3):539–549, 2010. doi: 10.1016/j.ejor.2009.07.031.

Kevin R. Gue and Russell D. Meller. Aisle configurations for unit-load warehouses. *IIE Transactions*, 41(3):171–182, 2009. doi: 10.1080/07408170802112726.

S. S. Heragu, L. Du, R. J. Mantel, and P. C. Schuur. Mathematical model for warehouse design and product allocation. *International Journal of Production Research*, 43(2):327–338, 2005. doi: 10.1080/00207540412331285841.

Chenyu Huang, Zhengyang Tang, Shixi Hu, Ruoqing Jiang, Xin Zheng, Dongdong Ge, Benyou Wang, and Zizhuo Wang. ORLM: A customizable framework in training large models for automated optimization modeling. *Operations Research*, 73(6):2986–3009, 2025. doi: 10.1287/opre.2024.1233. URL https://doi.org/10.1287/opre.2024.1233.

Interact Analysis. Warehousing sector forecast to grow 5% through 2027. https://interactanalysis.com/warehousing-sector-poised-for-5-growth-up-to-2027/, 2022.

Timo Looms and Lin Xie. Cpu-based layout design for picker-to-parts pallet warehouses, 2025. arXiv:2506.04266v2.

Theo Olausson, Alex Gu, Ben Lipkin, Cedegao Zhang, Armando Solar-Lezama, Joshua Tenenbaum, and Roger Levy. LINC: A neurosymbolic approach for logical reasoning by combining language models with first-order logic provers. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pp. 5153–5176. Association for Computational Linguistics, 2023. doi: 10.18653/v1/2023.emnlp-main.313. URL https://aclanthology.org/2023.emnlp-main.313/.

Ö. Öztürkoğlu, Kevin R. Gue, and Russell D. Meller. A constructive aisle design model for unit-load warehouses with multiple pickup and deposit points. *European Journal of Operational Research*, 236(1):382–394, 2014. doi: 10.1016/j.ejor.2013.12.023.

Liangming Pan, Alon Albalak, Xinyi Wang, and William Wang. Logic-LM: Empowering large language models with symbolic solvers for faithful logical reasoning. In *Findings of the Association for Computational Linguistics: EMNLP 2023*, pp. 3806–3824. Association for Computational Linguistics, 2023. doi: 10.18653/v1/2023.findings-emnlp.248. URL https://aclanthology.org/2023.findings-emnlp.248/.

B. Rouwenhorst, B. Reuter, V. Stockrahm, G. J. van Houtum, R. J. Mantel, and W. H. M. Zijm. Warehouse design and control: Framework and literature review. *European Journal of Operational Research*, 122(3):515–533, 2000. doi: 10.1016/S0377-2217(99)00020-X.

Atefeh Shahroudnejad, Payam Mousavi, Oleksii Perepelytsia, Sahir, David Staszak, Matthew E. Taylor, and Brent Bawel. A novel framework for automated warehouse layout generation. *Frontiers in Artificial Intelligence*, 7, 2024. doi: 10.3389/frai.2024.1465186.

A Formulation and Execution Details

This appendix specifies the modeling and execution contracts. Experiment-specific settings appear in Appendices B and C.1.

A.1 Geometry construction and clearance relations

The kernel receives storage and navigation polygons, obstacles, terminals, and a rectangular-rack catalog, in meters. With cell size h and origin at the domain bounding box's upper-left corner, cells wholly covered by the navigation domain define V_h; four-neighbor adjacency defines E_h. Rack candidates use grid anchors and fixed catalog dimensions, orientations, capacities, and service faces. Mandatory constraints enforce whole-footprint containment, non-overlap ($x_p + x_q \leq 1$ for conflicting candidates), and terminal preservation.

Each access width w induces a clearance graph for a disk of radius $w/2$. Complete node, edge, and connector geometries must clear boundaries, obstacles, and selected racks. For a statically admissible connection e with rack blockers B_e, availability is encoded by

$$a_e \leq 1 - x_p \quad (p \in B_e), \qquad a_e \geq 1 - \sum_{p \in B_e} x_p, \qquad a_e \in \{0, 1\}. \tag{9}$$

Traversal also requires open endpoints; source and service membership include connector checks. Footprint-overlap and clearance tolerances are 10^{-12} in area and 10^{-9} meters in distance. Default attachment radii are $h/\sqrt{2}$ for service ports and one meter for terminals. Refinement can change anchors and attachments, so feasibility need not be monotone even when centers are retained. Geometry changes invalidate dependent layouts, cuts, and bounds.

A.2 Formulation language and bounded lowering

The `warehouse-ip/0.1` schema binds finite placement, terminal, node, and edge sets to kernel-supplied geometry. Its sole authored binary family x_p selects racks. Available coefficients are catalog capacity (slots) and unit rack count; auxiliary variables are compiler-owned. Unit-checked linear arithmetic, comparisons, Boolean connectives, finite quantifiers, placement filters, containment, non-overlap, service-port, and reachability predicates form the permitted language. Selection-dependent predicates expand into finite templates; arbitrary products, code, and unbound parameters are rejected. Reachability denotes at least one active source–target pair; access from every terminal requires separate conditions.

Declarations cite requirement or assumption identifiers. Unknown references, incompatible units, unresolved requirements, and unsupported expressions block compilation. Reference coverage does not establish natural-language faithfulness. Operational obligations retain their simulation protocols. Bounded expansion produces auxiliaries z for $\mathcal{C}_{\text{phys}}(x) \wedge \mathcal{C}_{F_t}(x)$; CP-SAT may retain native Boolean constraints. Rational scaling is exact within declared numeric limits, and reported objectives and bounds reverse scaling and offsets. Unsupported scales or overflow raise errors. Inputs, bindings, geometry, and lowering artifacts are content-hashed.

A.3 Exact reachability and connectivity separation

Exact encoding. Let b_v, o_v, and r_v denote active source membership, availability, and reachability, respectively; $\rho_v \in \{0, \ldots, n\}$ is a rank over n potentially reachable nodes. The encoding imposes $b_v \Rightarrow r_v$, $r_v \Rightarrow o_v$, closure across active edges, and parent literals satisfying

$$y_{uv} \Rightarrow (r_u \wedge r_v \wedge a_{uv} \wedge \rho_v \geq \rho_u + 1), \qquad r_v \Rightarrow b_v \vee \bigvee_{u:\{u,v\} \in E_h} y_{uv}. \tag{10}$$

Source and unreachable ranks are zero. Closure marks every reachable node; conversely, decreasing parent ranks force each marked node to connect to a source. Target reachability is $\bigvee_v (r_v \wedge m_v)$ for active target membership m_v. This equivalence also supports negated reachability.

Connectivity cuts. The `separated_positive` strategy delays only complete top-level constraints of the form $\forall p \in P_h : x_p \Rightarrow \mathrm{Reach}(D', \mathrm{service}(p))$. Here D' is one terminal or the supplied terminal set. Every reachability occurrence must have this form; otherwise separation is ineligible and exact encoding requires authorized fallback. Replace undirected edges by paired arcs, attach a super-source s through terminal connectors, and attach each rack sink t_p through service connectors. Sinks have no outgoing arcs. Availability includes endpoint and connector conditions. For any S with $s \notin S$ and $t_p \in S$,

$$x_p \leq \sum_{e \in \delta^-(S)} a_e. \tag{11}$$

The cut contains *all potential incoming arcs*, including currently blocked ones. A valid source–sink path must enter S. Conversely, choosing S as the unreachable vertices separates any disconnected selected rack. The full cut family is therefore equivalent to the eligible positive access constraints.

An outer solve–check–cut loop adds violated cuts using an exact graph oracle. Candidates are released only after checking the original formulation. Certified master upper bounds remain valid for maximization, and a checked master optimum with recomputed objective proves static optimality. Bounds and infeasibility remain conditional on any active search restrictions or assignment exclusions.

SolvePlan contract. The plan binds the problem hash, chooses `exact_rank` or `separated_positive`, and specifies fallback, CP-SAT settings, and resource limits. Its profile provides structural counts, operators, objective sense, and separation eligibility. Plans cannot alter the formulation or supply unverified cuts. Compilation, solving, checking, separation, and fallback share a cooperative budget; candidates verified after the deadline are withheld. Model latency is charged separately when reporting total recorded cost. Strategy selection itself provides no acceleration guarantee.

A.4 Validation, revision, and stopping rules

The acceptance record fixes geometry, candidates, semantic reference, performance targets, and evaluation protocol:

$$\mathcal{A} = \big(\Omega, K, P_h, F^{\mathrm{acc}}, \mathcal{T}, W\big), \qquad \mathcal{T} = \{(j, \bowtie_j, \tau_j)\}_{j=1}^{m}. \tag{12}$$

The reference is independently reviewed when supplied, otherwise F_0; the latter makes acceptance conditional on the initial language interpretation. Declaration alignment is not a proof of semantic equivalence. An independent interpreter enforces the post-solve gate

$$g_t = \mathbb{I}[\mathcal{C}_{\mathrm{phys}}(x_t; \Omega, K) = 1 \ \wedge \ \mathcal{C}_{F^{\mathrm{acc}}}(x_t) = 1]\,. \tag{13}$$

Simulation additionally requires aligned declarations and remaining budget.

Writing $F_t = \mathrm{Compose}(\bar{F}_t, H_t)$, validated proposals update the semantic base or reversible search restrictions:

$$(\bar{F}_{t+1}, H_{t+1}) = \begin{cases} (\mathrm{Restore}_{F^{\mathrm{acc}}}(\bar{F}_t, \Delta_t), \varnothing), & \text{semantic correction,} \\ (\bar{F}_t, \mathrm{Hypothesis}(\Delta_t)), & \text{operational search.} \end{cases} \tag{14}$$

Corrections copy identified reference declarations; hypotheses replace recorded reversible restrictions without changing acceptance. Catalogs, bindings, thresholds, and workloads remain immutable. Proposals bind the current problem hash and cite feedback; repeated hypothesis cycles are rejected. Simulator failure does not certify a valid inequality. Optional assignment exclusion, $\sum_{p:\bar{x}_p=1}(1 - x_p) + \sum_{p:\bar{x}_p=0} x_p \geq 1$, removes only $\bar{x}$.

SimPy uses the supplied arrivals, fleet, service, routing, and dispatch model. The revision controller preserves disjoint demand classes, immutable weights, and reachable-capacity floors; it assigns orders to the nearest reachable rack within their class. This replenished class model differs from the fixed-layout comparison in Appendices B and C.1 and does not certify collision-free robot schedules. Throughput counts completions in the measurement window; cycle-time statistics include drain-period completions of orders arriving in that window.

A replication-level percentile bootstrap gives marginal 95% intervals $[\ell_{tj}, u_{tj}]$ for oriented margins $d_j(k_j(x_t; W) - \tau_j)$, where $d_j = 1$ for lower bounds and -1 for upper bounds. With complete,

Algorithm 1 Bounded acceptance and revision

Require: Initial formulation, fixed acceptance record, resource envelope
1: **for** each permitted iteration while budget remains **do**
2: Restore misaligned declarations; execute an eligible SolvePlan
3: If no candidate exists, restart with an admissible replan/revision or stop
4: Independently check the candidate; stop on checker disagreement
5: If no operational obligations remain, return static-only acceptance
6: Simulate the checked candidate under the search protocol
7: On PASS, return one final holdout verdict without further adaptation
8: On FAIL, apply an admissible revision or assignment exclusion; otherwise stop
9: **end for**

sufficiently sampled evaluations,

$$v_t = \begin{cases} \text{PASS}, & \ell_{tj} \geq 0 \quad \text{for every } j, \\ \text{FAIL}, & u_{tj} < 0 \quad \text{for some } j, \\ \text{INCONCLUSIVE}, & \text{otherwise.} \end{cases} \tag{15}$$

Demand-coverage and service failures also fail acceptance; invalid settings, unfinished orders, or insufficient samples prevent it. Intervals do not account for simulator bias or ensure joint metric coverage. Only search failures permit adaptation. The first search PASS triggers one final evaluation on predeclared distinct random streams; its outcome is never fed back. Inconclusive evaluations, errors, timeouts, and proven finite infeasibility remain distinct outcomes. The controller budget charges model calls, revision, compilation, solving, and both evaluations.

A.5 Software and library resource envelopes

The implementation uses Python 3.12.13, warehouse-formulation 0.6.0, OR-Tools 9.15.6755, SimPy 4.1.1, Shapely 2.1.2, and Pydantic 2.13.4. Configurable library defaults are a 30-second solve limit, 60-second compilation and plan limits, one CP-SAT worker, and 32 separation rounds. The controller defaults to six iterations within 120 seconds, at most three revision calls, and two plan calls plus two replans. Simulation defaults to at least three replications and 30 cycle samples, 500 bootstrap resamples, and a 60-second limit. These API defaults are distinct from the empirical protocols in Appendices B and C.1; serialized configurations and bounded compiler limits determine each run.

B Explicit-Formulation Main Experiment

Instantiated pipeline. The uniform-0.33 main experiment instantiates FLOWS with 42 archived LLM-generated IP formulations, a fixed constructive portfolio, exact feasibility screening on its candidate subset, and the full-formulation interpreter. The catalog has three lengths derived from historical base dimensions, five levels, and declared capacities of 10/20/30 units. Each inherited formulation used one `deepseek-v4-pro` request with temperature zero, a 16,384-token output limit, a 120-second socket timeout, and no retry; generation costs are charged once.

Representation and construction. The compiler uses grid spacing $h = 2c/3$ for base rack width c, safe nodes, and four-neighbor edges. Portfolio service attachments are augmented within $1.5h$ using the original exact clearance predicate, preserving the 0.8-m passage requirement and rebinding the geometry to the formulation. This improves sampling coverage without proving continuous-space completeness. Paired rack banks leave longitudinal and periodic transverse passages aligned to safe half-integer coordinates. Bank lengths are 2, 4, and 8 units for $c \geq 1$ m, and 4, 8, 16, and 2 otherwise; each uses phases 0/2 and orientations $0°/90°$. Recipes add the longest catalog racks in depot-distance and candidate-identifier order until the 0.33 target or pool exhaustion. Batch screening preserves sequential-addition feasibility in the supported monotone fragment; retained checkpoints also pass the full interpreter and physical/development checks. Selection maximizes development throughput among target-reaching candidates, then minimizes cycle time and travel; otherwise it prioritizes utilization, then throughput. No baseline layout or KPI enters selection, and portfolio failure does not certify IP infeasibility.

Budget and cohorts. The 300-second local budget includes compilation, service augmentation, serialization, feasibility-cache construction, candidate generation, symbolic checks, and development simulation. The interpreter has a fixed ten-million-operation allowance; only checkpoints verified before the deadline are eligible. Initialization/source verification, final publication, and generation/confirmation are outside this timer. The final cohort combines disjoint 15- and 27-case batches with matching configuration and 42 solver-source hashes. Earlier 0.32 trials and supplements remain in the development ledger, including failures; the final uniform-0.33 setting was motivated by density shortfalls, not confirmation-KPI selection. C03, C20, and C36 remain below 0.32 under the fixed portfolio, with C03 limited by disconnected narrow passages in the coarse graph. The GPT6 comparison separately uses a 0.34 construction target. All cases were development-exposed and density follow-ups reuse confirmation streams, so target selection is not independently held out.

Evaluation and capacity. Development uses three streams and confirmation uses 20 disjoint streams, once after locking each layout, reference geometry, and profile. Fixed comparators share confirmation streams and matched arrival counts. Availability requires common output, dimension, geometry, access, and completed-replication checks; it does not certify a service-level target. The 120-orders/h load exceeds achieved throughput, making this a 24-hour overload test rather than a steady-state analysis. Native capacity is unmatched: CBS counts area-based grid positions at one level, CPU counts bank length/service faces/levels, and FLOWS uses declared catalog units. Destination sampling uses these native position counts after removing method-specific demand weights, so rack partitioning and relative capacity weights can affect travel and congestion. The post-hoc inventory-occupancy field is $\min\{S, \lfloor 120 \times 18 + 3N_{\mathrm{rack}} \rfloor\}/S$; neither it nor native positions supports cross-method storage-gain claims.

Main-experiment runtime. Table 4 weights all 42 cases equally. Generation includes prompt construction and response parsing; planning includes compilation and development simulation; confirmation includes physical checks and all 20 replications. The 49.31-second mean compilation/serialization time and 9.68-second feasibility-cache initialization are already included in the 168.44-second planning mean. All planning runs finish within 300 seconds (maximum 297.60), and selected checkpoints meet the deadline. The 291.07-second component sum (4.85 minutes) is descriptive accounting, not measured end-to-end latency: generation is historical and worker initialization, source verification, outer-process overhead, and final publication are excluded.

Table 4: FLOWS runtime distribution across 42 main cases, in seconds. SD is sample standard deviation; P90 uses linear interpolation. Component sums are calculated per case before aggregation.

Component	Mean	SD	Median	P90	Min–max
Formulation generation	111.47	26.56	112.89	143.75	57.60–177.19
Local planning	168.44	39.58	160.45	226.71	81.04–297.60
Confirmation (20 replications)	11.16	2.18	11.28	13.13	4.77–16.28
Recorded component sum	291.07	45.36	286.70	352.17	208.10–401.75

Baseline planning measures planner calls after recognition, excluding final checks and simulation; FLOWS additionally compiles and evaluates candidates in simulation. CBS planning coverage is 40/42: C03/C17 lack separable complete recovery timers, and C40 excludes subsequent unreachable-rack pruning. Missing values remain missing; mixed recovery-worker timers are not substituted. CPU planning and all methods' confirmation timings cover 42/42 cases. Different timing boundaries and host contention preclude controlled cross-method speedup claims.

Development cost and evidence. Main planning, confirmation, and inherited generation sum to 7074.60, 468.69, and 4681.65 seconds, respectively. The original 42 calls use 503,723 tokens (193,391 prompt plus 310,332 completion, including reasoning), inherited once across configurations. The density follow-up ledger retains 103 attempts, 16385.49 seconds of local planning, and 4,300 confirmation replications, including comparators; development simulation is additional and already included in planning. Earlier diagnostics remain in their archives outside this ledger. Two planning workers and one confirmation worker overlap, so batch wall times are not additive. Source hashes, layout/profile bindings, arrival counts, raw replication means, and the disjoint cohort partition are

checked for artifact consistency, without claiming semantic completeness or a full execution-history proof.

Exploratory paired uncertainty. We bootstrap warehouse pairs 20,000 times using each layout's 20-replication mean. Percentile 95% throughput-difference intervals (FLOWS minus comparator, orders/h) are $[-0.634, 5.360]$ for CBS, $[4.040, 8.715]$ for CPU, $[1.223, 14.414]$ for DeepSeek, and $[0.458, 3.976]$ for GPT6 on their respective cohorts. CBS's interval includes zero; utilization intervals include zero against CBS and both direct generators. These exploratory intervals condition on development-exposed cases and fixed layouts, omit generation/target-selection variability, and have no multiple-comparison adjustment.

Descriptive operating diagnostics. The analysis recomputes the 126 saved confirmations without new simulation or selection. For completed-order case means, service time is $T_{\text{cycle}} - T_{\text{queue}}$ and travel time is $D/(60v)$ minutes, where D is round-trip distance and $v = 1.6$ m/s. FLOWS/CBS/CPU mean service times are 4.434/4.757/4.946 minutes; travel contributes 1.540/1.833/1.996 and edge blocking 0.091/0.120/0.145 minutes. Travel arithmetically accounts for 90.7%/89.3% of the service-time differences against CBS/CPU; these are not causal effects. Mean AGV utilization is 99.920%/99.903%/99.901%, while allocation waiting accounts for about 98.5% of cycle time. Travel savings correlate descriptively with throughput gains ($r = 0.882/0.958$ against CBS/CPU). Against CBS, the 21 nonlower-utilization cases average -0.1015 orders/h, versus $+5.5536$ for the 21 lower-utilization cases; all cases, including minimal-storage baselines, remain included. Reported travel improvements compare group means, not mean casewise percentages. Unmatched destination weights and capacities prevent attributing these associations solely to simulation-based selection.

C Evaluation Protocol and Comparator Provenance

The main and comparator-floor experiments share a capacity-weighted, finite-horizon evaluator. It measures layout availability and operating KPIs, distinct from the controller's class-based, target-aware acceptance protocol.

C.1 Geometry, Physical Checks, and Simulation

Shared geometry and eligibility. All methods use the same 42 annotated storage polygons, holes, obstacles, and obstacle-aware southwest depots. C01's millimeter interpretation is a shared drawing-based scale hypothesis, not independently verified ground truth. Eligibility requires nonempty output, finite positive dimensions, dimensional limits, containment, collision avoidance, and physical depot access to every rack. The common limits are 86 m for the longer footprint side, 2.5 m for the shorter side, and 8.5 m for height, regardless of rotation. Navigation accounts for boundaries, holes, obstacles, racks, service faces, and 0.4-m center clearance for a 0.8-m-wide AGV. Paths are shortest on the constructed graph, without a continuous-space optimality guarantee or route-graph fallback. Reevaluation never moves, resizes, or deletes racks to rescue an output; archived CBS adaptations are described below.

Workload and measurement. Poisson arrivals offer 120 orders/h to five FCFS-allocated AGVs moving at 1.6 m/s. Destination racks are sampled in proportion to native position counts. Each vehicle remains allocated for the depot round trip, route-edge waiting, and handling, with handling time $\max\{1.2, \mathcal{N}(2.8, 0.55^2)\}$ minutes. Edges have capacity one and a 4-second interval between reservations by different orders; this approximation omits collision-free multi-robot scheduling, turning dynamics, and inventory depletion.

Replications stop at 24 hours without warm-up or draining. Throughput is completions divided by 24 hours; cycle time includes AGV-allocation and edge waiting. Cycle, P90 cycle, edge blocking, and round-trip travel use completed orders only, while incomplete counts remain recorded. Edge blocking excludes AGV-allocation waiting. Utilization divides total rack footprint by storage-polygon area after subtracting holes, without subtracting obstacles again. Each KPI averages 20 replication-level values per case, then weights cases equally; reported P90 is a mean of replication percentiles. Handling alone limits throughput to approximately 107 orders/h, so this is an overload test, not steady-state delay

estimation. Completing confirmation establishes availability, not service-level or uncertainty-aware acceptance.

C.2 Fixed Planning Baselines

CBS-24 and CPU-PES-25. CBS adapts constrained beam search and Pareto scoring (Shahroudnejad et al., 2024) to polygons, reserved access, raster exclusions, and bank clearances. CPU adapts P/E/S zoning (Looms & Xie, 2025) with terminal/apron clearances and first-feasible orientations $0°$, $90°$, $180°$, $270°$, without confirmation-KPI selection. The comparison evaluates layout adaptations, not the original CPU study's complete ABC-slotting and inbound/outbound policies. Both adaptations were developed on these 42 inputs; current evaluation reuses fixed layouts.

CBS minimum-output recovery. The fixed cohort includes deterministic recovery for otherwise empty C03/C17 outputs: adjacent available-cell pairs on the existing 900-cell grid define single-rack candidates, ordered by decreasing footprint and then coordinates/dimensions. The first candidate passing common physical checks is locked, yielding one rack per case and raising availability from 40/42 to 42/42. This extension uses no LLM, refinement, depot relocation, or KPI feedback and need not satisfy the original grid filters; it is not attributed to the CBS paper. C40 separately retains 41 of 43 racks after archived unreachable-rack pruning. These construction adaptations are not repeated during confirmation. Minimal nonempty outputs establish physical availability, not adequate storage, and can obtain favorable travel/throughput under capacity-weighted demand.

C.3 Direct-Generation Protocol and Current Re-evaluation

Both direct generators receive anonymized numeric geometry, depot, operating conditions, and rack catalog, then return axis-aligned rectangles as JSON. Each has one attempt without previous layouts, KPI feedback, repair, or evaluation seeds. Archived outputs undergo the same locked physical checks and confirmation as the main experiment.

DeepSeek. Each independent, tool-free request enables thinking, temperature zero, and a 131,072-token output cap including reasoning. The requested `deepseek-v4-flash` returns `deepseek-flash`; it is not model-version-matched to formulation generation. The reported 42-case run excludes an earlier low-cap diagnostic and performs no retry or cross-run selection. All outputs satisfy the generation schema within 300 seconds; common checks first reject four for dimensions, 15 for geometry, and eight for reachability, leaving 15 confirmed layouts.

GPT6. The archived native-model experiment uses 42 independent history-free contexts with the same design messages and instructed restrictions against computational design tools or feedback; file I/O may span turns. Exact model snapshot, sampling settings, output cap, and complete traces are unavailable, so tool compliance is not independently verified. The cutoff was amended during generation and applied uniformly as 300 seconds from recorded dispatch to first saved JSON. Dispatch logging for C01–C23 can understate latency. Fifteen timely outputs pass confirmation; 27 timeouts comprise 15 late saved outputs and 12 without saved JSON. Late outputs remain ineligible.

Locked cohorts and cost. All 84 outcomes remain recorded; eligible layouts are hash-verified and locked without correction, fallback, or selection among outputs. The new 20-stream evaluation measures fixed-layout simulation variability, not independent generation variability. Table 3 conditions on each generator's 15 available cases; FLOWS uses uniform-0.33 layouts against DeepSeek and separate 0.34 layouts against GPT6. The DeepSeek cases are C01, C06, C07, C10, C14, C15, C16, C21, C22, C31, C33, C37, C40, C41, C42; GPT6 uses C07, C10, C14, C15, C17, C18, C19, C23, C26, C28, C36, C37, C40, C41, C42. These development-exposed cohorts do not match utilization casewise or provide held-out target selection.

DeepSeek uses 624,851 tokens on its 15-case cohort (41,656.73/case) and 2,095,901 across 42 attempts (49,902.40/attempt), excluding the discarded diagnostic and counting cached input/reasoning once. GPT6's 15 timely contexts use 1,356,892 input and 116,059 output tokens (98,196.73/case), including repeated context, file I/O, and acknowledgement but excluding parent orchestration and simulation. Incomplete usage and post-cutoff consumption prevent an exact 42-attempt budget-censored mean. FLOWS's inherited means are 11,069.13/12,051.53 tokens on these cohorts; replay

and reevaluation use no new model tokens. Unequal scopes support workflow accounting, not model-level or equal-budget efficiency claims.

Runtime components. Table 5 reports the same cohorts. FLOWS generation lies outside its 300-second planning budget, while direct-generation limits cover generation. Confirmation excludes source verification and outer-process overhead; host contention also differs. Component sums are descriptive accounting under the main-cohort timing conventions.

Table 5: Mean runtime components on direct-generation cohorts, in seconds. Generation is archived; planning and confirmation are separately observed. Dashes denote no local planning stage; sums use unrounded values.

Cohort	Method	Generation	Local planning	Confirmation	Component sum
DeepSeek, $n = 15$	DeepSeek	135.43	–	4.75	140.18
	FLOWS	104.56	178.24	11.25	294.05
GPT6, $n = 15$	GPT6	246.58	–	6.02	252.61
	FLOWS	109.75	164.06	11.69	285.51

C.4 Three-Method Comparison on the Shared Eight Cases

Table 6 uses C07, C10, C14, C15, C37, C40, C41, and C42 with common confirmation streams. Both FLOWS targets improve mean operational KPIs, while direct generators remain denser on this success-conditioned intersection; the comparison supplements full-cohort availability and pairwise results.

Table 6: Shared-eight-case comparison with 20 common confirmation streams. FLOWS subscripts denote targets; bold marks best KPIs, not significance. Token scopes follow Table 3.

Method	Util. ↑ (ratio)	Throughput ↑ (orders/h)	Mean cycle ↓ (min)	P90 cycle ↓ (min)	Edge block ↓ (min)	Travel ↓ (m)	Tokens (k/case)
DeepSeek	**0.3498**	65.37	327.16	585.78	0.148	183.52	38.87
GPT6	0.3457	72.07	286.77	513.24	0.101	122.00	89.83
FLOWS$_{.33}$	0.3304	**74.93**	**269.42**	**481.72**	**0.070**	**109.57**	10.48
FLOWS$_{.34}$	0.3403	73.97	275.20	492.10	0.078	114.06	10.48

D Comparator-Specific Utilization Floors

For warehouse i and comparator m, the fixed layout L_{im}^{base} sets FLOWS's utilization floor:

$$u_{im}^{\min} = \frac{\sum_{r \in L_{im}^{\text{base}}} w_r d_r}{A_i}, \qquad U(L_{im}^{\text{FLOWS}}) \geq u_{im}^{\min}. \tag{16}$$

Here A_i is storage-polygon area excluding holes, without subtracting obstacles again. Construction and evaluation use full-precision geometry with 10^{-12} comparison tolerance and unchanged physical references. The floor prevents accepting lower utilization; it does not match standardized capacity or destinations.

Construction and confirmation. The 114 tasks comprise 42 each against CBS/CPU and 15 each against the available direct generators; the other 54 direct outcomes retain failure/timeout status. Each task uses the inherited formulation and fixed portfolio (Appendix B), with only the comparator's utilization scalar entering planning. Coordinates, routes, and KPIs are never supplied, and main-experiment layouts remain unchanged. The 300-second budget includes compilation, construction, checks, and development evaluation, excluding generation and confirmation. At most four task processes run concurrently, each with one BLAS/OpenMP thread; these runs do not compare planning speed with archived batches. The floor applies to whole-layout qualification, preserving the incremental oracle's downward-closed screening semantics. Selection maximizes development

throughput among qualifying candidates, breaking ties by cycle time and travel. If none qualifies, the best below-floor incumbent remains diagnostic; no threshold relaxation, model call, or best-of-retry selection occurs.

Three development streams and 20 disjoint confirmation streams are also distinct from earlier experiments. Layouts are locked before confirmation; comparators are reevaluated on the same streams under the common workload and capacity-weighted destination rule, with arrival counts matched. Confirmation cannot affect selection. Hashes bind inputs, targets, code, and outputs, and all attempted statuses are retained.

Failures and interpretation. Seven saved below-floor incumbents subsequently receive diagnostic simulation on the same 20 confirmation streams, without replanning, repair, or selection. These measurements supply failure points only; thresholds, statuses, and successful-pair summaries are unchanged. Summaries weight warehouses equally and condition on physical availability and floor attainment, reporting qualification counts over every eligible cohort and utilization shortfalls/excesses. Exploratory intervals resample paired warehouses, not simulator replications. The new streams limit stochastic outcome reuse, but development-exposed cases and comparator outputs preclude unseen-facility or isolated-component claims.

Observed outcomes. Table 7 separates utilization-floor attainment from successful paired confirmation. Each denominator includes every predeclared eligible task; the original comparator denominator is 42 inputs per method. CBS and CPU each have 42 eligible outputs; each direct generator has 15, with its other 27 historical outcomes excluded under the original eligibility rules. Operating differences, win counts, and utilization excess condition on floor-qualified pairs and therefore do not replace full-cohort availability accounting.

Table 7: Results under comparator-specific utilization floors. Floor and paired counts use each complete eligible cohort. ΔTP is the paired mean throughput difference in orders/h (FLOWS minus comparator), followed by its exploratory 95% interval. W/T/L counts throughput wins/ties/losses; $\overline{\Delta U}$ is mean utilization excess in percentage points. Differences and excess use successful pairs only.

Comparator	Floor	Paired	ΔTP [95% CI]	W/T/L	$\overline{\Delta U}$ (pp)
CBS	39/42	39/42	$+5.15$ $[+3.246, +6.920]$	33/0/6	0.0368
CPU	42/42	42/42	$+16.58$ $[+14.681, +18.757]$	42/0/0	0.0402
DeepSeek	11/15	11/15	$+8.29$ $[+2.511, +14.493]$	7/0/4	0.0477
GPT6	15/15	15/15	$+3.70$ $[+1.304, +6.291]$	10/0/5	0.0384

The intervals use 20,000 percentile bootstrap resamples of paired warehouse-level throughput differences, conditional on the successful layouts and development-exposed warehouses. They do not include variation from model generation or target selection and are not adjusted for multiple comparisons.

Shortfalls and realized utilization. All seven failures complete the prescribed portfolio within 300 seconds, with no screening or verification timeout across 92 construction attempts; they do not establish physical infeasibility. Below-floor incumbents remain diagnostic outputs and are never counted as satisfying the original lower bound. Table 8 lists all seven failures, including their original utilization shortfalls and additional diagnostic throughput. Figure 6 shows attained utilization alongside the imposed floor for all 114 eligible tasks.

E Component Ablation Protocols and Results

These frozen component experiments use two separate 12-case synthetic suites, not a 42-case end-to-end ablation. Local repeats reproduce symbolic execution and offline feedback without new model calls or independent samples; tables retain original timings and actual call costs.

Table 8: All seven unsuccessful comparator-floor tasks. Utilizations and shortfalls retain their original values. Throughput (TP, orders/h) averages 20 paired streams; FLOWS TP comes from additional diagnostic simulations of the unchanged below-floor layouts. These rows remain failures and do not enter the successful-pair means.

Comparator	Case	Floor (%)	Attained (%)	Shortfall (pp)	Baseline TP	FLOWS diagnostic TP
CBS	C28	51.7444	49.4577	2.2866	43.47	50.25
CBS	C31	51.0937	50.3088	0.7848	54.56	52.00
CBS	C40	40.7953	38.4869	2.3084	64.50	61.99
DeepSeek	C14	45.4737	41.9956	3.4781	39.65	75.20
DeepSeek	C31	60.2926	50.3088	9.9838	57.49	52.00
DeepSeek	C41	36.4273	35.5733	0.8540	47.91	72.62
DeepSeek	C42	51.0923	39.6855	11.4069	79.99	59.78

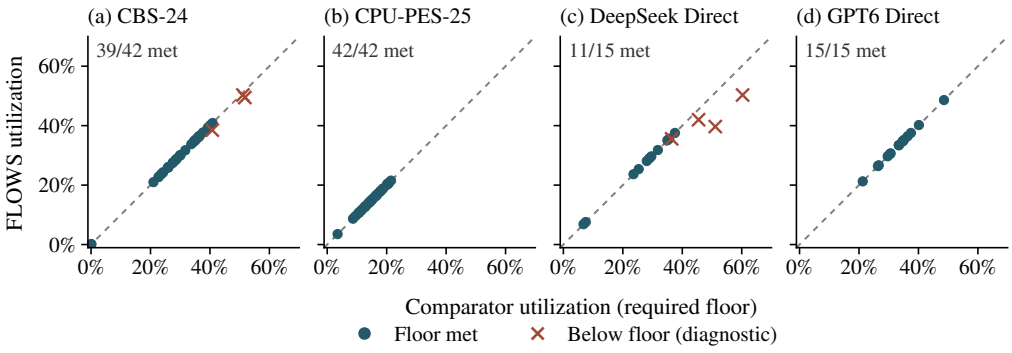

Figure 6: Required and attained footprint utilization for all 114 eligible tasks. The diagonal denotes exact equality with the comparator. Teal circles mark the 107 floor-qualified successful pairs; red crosses mark all seven below-floor layouts, whose diagnostic throughput does not change their failure status.

E.1 Removing the explicit symbolic representation

The direct generators receive numeric polygons, holes, obstacles, dock, catalog, and requirements, then emit rack rectangles without a graph, IP, symbolic search, or repair. Thus this architectural control jointly removes several components; it is neither image-only nor an isolated representation ablation. The unchanged external evaluator counts invalid/late outputs as failures: each generator provides 15/42 available layouts versus FLOWS's 42/42, under the direct-generation protocol above.

E.2 Removing solver-strategy selection

We fix authored formulations across three variants of four geometries: rectangles, L-shapes, central obstacles, and narrow necks. Controls use fixed exact encoding, fixed separation, or a deterministic rule selecting exact encoding at these sizes. The adaptive arm makes one `deepseek-v4-pro` SolvePlan call per case, reused across three paired solver trajectories with rotated method order. All arms allow two seconds per native solve, five for compilation, eight for plan execution, one worker, eight separation rounds, and 256 cuts. Plan generation separately allows 240 seconds and 32,768 output tokens without retry. Table 9 takes the median of casewise trajectory medians; end-to-end time includes initial compilation, plan generation, and execution, excluding artifact writing and the fixed formulation's generation.

Table 9: Strategy ablation on 12 synthetic instances. All paired trajectories reach checked optima with identical objectives. Times are medians of casewise medians.

Procedure	Optimal cases	Execution (ms)	End-to-end (s)	LLM calls
Fixed exact encoding	12/12	27.4	0.032	0
Fixed separation	12/12	87.6	0.095	0
LLM-selected plan	12/12	86.0	37.572	12

All 12 plans choose separation, matching its selected layouts, with exact fallback in 24/36 executions. The 12 calls use 63,534 tokens, charged once per reused plan. Fixed exact solving is fastest; these instances support interface correctness, not LLM-driven acceleration.

E.3 Removing validation feedback and formulation revision

Six declaration-error cases, four operational cases, and nominal/infeasible controls share formulations, exact strategy, acceptance references, workload, and simulation. No feedback permits one iteration, zero revisions, and no assignment exclusions, while retaining all acceptance checks and final confirmation. Iterative arms allow three iterations and one revision within 300 seconds: exclusion changes assignments only; reference restoration deterministically repairs declarations from the same reference visible to the LLM. No arm invokes the strategy agent.

Table 10: Feedback ablation on a separate 12-case repair suite. Counts require final acceptance; semantic/operational columns identify case categories.

Procedure	Accepted	Semantic	Operational	Controls	LLM calls
No feedback	4/12	0/6	3/4	1/2	0
Assignment exclusion	4/12	0/6	3/4	1/2	0
Reference restoration	9/12	5/6	3/4	1/2	0
LLM revision	9/12	5/6	3/4	1/2	7

Seven `deepseek-v4-pro` calls use 36,701 tokens, including one rejected infeasible-control repair. Six restore declarations; five reach acceptance and one remains inconclusive at confirmation. All operational cases initially pass search simulation, so none triggers KPI revision; one is later inconclusive. The nominal control passes and the infeasible control fails under every procedure. LLM and deterministic restoration produce identical casewise statuses, layouts, and KPIs. The 5/12 gain therefore measures recovery of known declaration errors, not superior operational reasoning. Unequal iteration budgets and these synthetic errors leave benefits on natural errors, actual KPI failures, and the main cohort unestablished.

F Complete Main-Experiment Results

Table 11 reports all 126 case–method results in the uniform-0.33 main comparison, including every below-target FLOWS layout. CBS and CPU denote the adapted CBS-24 and CPU-PES-25 baselines. All layouts pass the common physical checks; operating KPIs average 20 paired confirmation replications per layout. U is footprint utilization; T throughput; C mean cycle time; C_{90} P90 cycle time; B route-edge blocking; and D mean travel. G, P, and V are recorded formulation-generation, planning, and final confirmation times. FLOWS's P includes compilation, checking, and development simulation; baseline P records planner calls. Generation is inapplicable to the two non-LLM baselines. CBS C03/C17 have no separate complete recovery-planning timer (–), and C40's P excludes subsequent pruning (‡); the CBS mean P therefore covers 40 cases (†). All other means cover 42 cases. These runtime components do not constitute controlled end-to-end latency measurements.

Table 11: All main-experiment cases, KPIs, and runtime components. Case means and complete coverage are retained; dashes and timing qualifications are defined above.

Case	Method	U (%)	T (orders/h)	C (min)	C_{90} (min)	B (min)	D (m)	G (s)	P (s)	V (s)
C01	CBS	28.54	63.00	340.55	611.34	0.121	174.99	–	28.80	3.24
	CPU	12.89	66.15	322.04	578.29	0.094	155.93	–	0.51	3.56
	FLOWS	33.09	63.86	336.37	604.41	0.116	169.50	97.33	150.78	12.33
C02	CBS	37.64	57.41	374.30	670.33	0.107	220.77	–	21.82	2.96
	CPU	16.66	55.06	389.43	696.87	0.133	239.53	–	0.33	3.98
	FLOWS	33.00	65.92	324.24	582.30	0.082	158.59	104.80	130.72	8.68
C03	CBS	0.09	90.23	176.60	315.11	0.150	34.94	–	–	1.21
	CPU	3.49	38.40	489.56	876.43	0.389	440.44	–	0.62	1.91
	FLOWS	5.74	80.35	236.33	421.82	0.089	80.13	122.29	81.04	4.77
C04	CBS	28.85	58.67	366.81	658.50	0.130	207.78	–	40.90	3.35
	CPU	12.08	63.14	339.92	611.38	0.115	174.69	–	0.79	2.68
	FLOWS	33.00	69.01	305.13	546.66	0.093	138.03	126.95	133.08	7.70
C05	CBS	23.58	77.92	250.68	448.92	0.072	92.54	–	28.18	2.31
	CPU	12.53	73.69	276.25	494.61	0.102	110.83	–	1.08	2.93
	FLOWS	33.00	78.28	249.09	444.62	0.086	89.66	113.64	141.30	11.33
C06	CBS	34.90	45.60	446.58	799.68	0.160	344.69	–	21.22	6.40
	CPU	14.75	38.51	489.50	876.39	0.461	431.39	–	1.30	3.74
	FLOWS	33.04	48.38	429.51	770.50	0.121	312.46	86.56	260.12	12.46
C07	CBS	40.88	64.49	331.01	595.77	0.075	168.85	–	23.89	1.96
	CPU	15.01	64.60	331.21	595.63	0.225	153.89	–	0.66	2.61
	FLOWS	33.00	74.16	273.85	489.68	0.082	110.30	140.78	181.30	11.70
C08	CBS	35.84	64.09	333.51	600.07	0.112	168.06	–	26.90	3.67
	CPU	21.47	65.61	325.15	583.97	0.101	158.66	–	0.99	4.15
	FLOWS	33.08	72.55	283.58	507.69	0.060	120.98	140.41	248.23	13.11
C09	CBS	35.03	67.45	313.85	563.05	0.089	148.19	–	44.68	2.89
	CPU	20.38	63.84	335.47	603.39	0.081	172.75	–	0.86	3.75
	FLOWS	33.05	72.16	286.04	512.37	0.063	122.82	107.90	231.91	10.07
C10	CBS	35.94	76.45	259.75	465.38	0.089	98.16	–	32.20	2.95
	CPU	15.33	69.41	302.12	541.21	0.100	135.15	–	0.51	3.72
	FLOWS	33.00	76.50	260.02	464.10	0.083	98.39	82.93	148.37	14.51
C11	CBS	36.62	59.77	359.93	645.49	0.199	192.17	–	36.67	2.07
	CPU	20.71	54.70	390.96	701.29	0.265	230.25	–	0.57	3.86
	FLOWS	33.08	72.86	281.74	504.21	0.055	119.88	95.19	157.88	9.08
C12	CBS	26.00	63.12	339.21	610.85	0.115	174.69	–	44.33	3.92
	CPU	9.44	63.39	338.27	607.98	0.113	172.93	–	0.77	4.18
	FLOWS	32.41	62.18	346.89	622.05	0.130	180.15	98.80	140.13	12.19
C13	CBS	36.29	69.97	298.96	534.72	0.113	130.63	–	30.13	3.02
	CPU	13.88	64.96	328.75	590.83	0.126	160.84	–	0.73	3.07
	FLOWS	33.08	70.09	298.61	533.78	0.095	131.56	149.04	136.23	10.17

Continued on the next page

Table 11 (continued)

Case	Method	U (%)	T (orders/h)	C (min)	C_{90} (min)	B (min)	D (m)	G (s)	P (s)	V (s)
C14	CBS	34.50	83.79	215.85	384.69	0.058	68.51	–	19.86	1.86
	CPU	12.85	73.64	276.63	495.31	0.157	105.87	–	0.32	2.28
	FLOWS	33.08	80.74	234.29	418.08	0.050	82.15	128.47	185.94	11.66
C15	CBS	26.02	63.03	341.13	612.08	0.115	175.57	–	65.59	3.09
	CPU	14.80	63.23	339.71	610.20	0.110	174.59	–	0.63	2.99
	FLOWS	33.08	67.91	312.17	559.37	0.071	146.85	98.45	141.09	11.53
C16	CBS	40.15	47.14	438.31	784.46	0.243	316.29	–	24.22	2.88
	CPU	14.15	44.47	452.69	811.30	0.196	357.41	–	0.65	3.57
	FLOWS	33.00	60.90	354.08	634.49	0.098	192.99	137.11	177.59	10.18
C17	CBS	0.08	102.93	100.12	177.76	0.033	6.85	–	–	1.42
	CPU	8.68	65.15	329.06	590.12	0.121	160.09	–	0.73	3.99
	FLOWS	33.02	54.84	390.54	698.79	0.185	236.62	123.63	199.95	9.96
C18	CBS	25.78	80.53	235.03	419.75	0.070	81.04	–	38.96	2.78
	CPU	11.54	77.30	254.61	455.58	0.110	92.13	–	0.47	2.89
	FLOWS	33.09	78.90	245.18	437.47	0.091	86.22	114.17	177.97	12.70
C19	CBS	36.36	72.68	282.26	506.45	0.091	117.31	–	47.72	2.63
	CPU	18.62	69.21	303.43	543.61	0.095	136.81	–	0.63	3.19
	FLOWS	33.00	73.60	276.98	495.57	0.089	112.65	66.47	141.97	11.16
C20	CBS	21.00	65.00	328.74	589.82	0.084	164.39	–	62.94	4.79
	CPU	14.22	62.00	346.75	622.98	0.120	182.30	–	0.94	4.19
	FLOWS	31.97	63.94	335.74	603.24	0.118	168.73	107.94	165.61	11.89
C21	CBS	27.41	31.21	534.94	953.92	0.140	635.68	–	56.37	5.68
	CPU	15.92	41.89	467.99	838.35	0.150	401.29	–	1.59	6.21
	FLOWS	33.01	45.33	448.45	802.06	0.105	353.80	69.53	297.60	15.81
C22	CBS	33.73	48.11	431.64	773.83	0.302	298.36	–	37.77	2.71
	CPU	17.17	44.44	453.82	812.82	0.132	364.05	–	1.37	3.95
	FLOWS	33.00	56.30	381.97	683.18	0.114	229.85	128.08	177.21	10.40
C23	CBS	36.34	74.89	269.23	481.54	0.061	108.60	–	34.59	2.46
	CPU	17.85	72.12	285.95	512.50	0.104	119.15	–	0.47	3.40
	FLOWS	33.00	78.23	249.39	445.31	0.058	92.57	107.33	156.94	11.89
C24	CBS	27.56	58.74	366.01	656.56	0.137	206.59	–	36.30	3.25
	CPU	16.45	60.63	355.04	636.93	0.108	194.09	–	0.67	3.64
	FLOWS	33.08	59.00	365.05	654.53	0.146	203.47	143.90	121.99	11.43
C25	CBS	31.70	55.70	385.06	689.98	0.156	231.38	–	56.50	3.30
	CPU	17.73	59.79	360.10	645.59	0.162	195.50	–	0.91	4.38
	FLOWS	33.00	62.87	342.34	614.56	0.127	175.52	142.42	155.35	12.50
C26	CBS	26.03	76.04	262.59	468.93	0.079	101.19	–	29.20	3.66
	CPU	16.24	70.40	296.50	530.26	0.137	125.73	–	0.64	2.78
	FLOWS	33.08	79.60	241.22	429.85	0.057	86.49	115.30	229.03	13.99
C27	CBS	24.06	77.04	255.79	458.03	0.103	93.97	–	49.25	2.45
	CPU	14.16	77.80	251.40	450.19	0.080	92.61	–	0.52	3.22
	FLOWS	33.08	80.97	232.84	415.80	0.061	79.84	87.00	194.32	13.13
C28	CBS	51.74	43.48	459.05	822.18	0.396	352.67	–	38.50	2.24
	CPU	20.24	51.09	413.04	740.86	0.168	276.85	–	0.99	4.92
	FLOWS	33.00	59.42	362.73	649.86	0.092	205.03	177.19	164.54	10.69
C29	CBS	39.24	64.65	330.12	593.51	0.088	166.56	–	30.17	4.04
	CPU	20.74	60.61	354.92	636.85	0.110	193.73	–	0.90	4.97
	FLOWS	33.08	71.10	292.59	523.34	0.060	129.21	79.39	168.56	10.80
C30	CBS	35.55	73.24	279.38	499.56	0.085	114.78	–	55.15	2.98
	CPU	12.65	70.88	293.25	525.58	0.210	116.02	–	0.66	2.51
	FLOWS	33.08	75.18	268.05	478.73	0.061	107.19	95.98	129.33	10.13
C31	CBS	51.09	54.78	390.95	700.21	0.093	245.89	–	52.88	3.06
	CPU	20.31	52.36	405.52	727.39	0.206	259.28	–	0.79	4.15
	FLOWS	33.08	62.15	346.70	622.17	0.099	183.55	98.18	163.15	9.63

Continued on the next page

Table 11 (continued)

Case	Method	U (%)	T (orders/h)	C (min)	C_{90} (min)	B (min)	D (m)	G (s)	P (s)	V (s)
C32	CBS	30.04	63.12	339.96	610.93	0.089	177.36	–	49.39	3.75
	CPU	10.45	64.71	330.92	594.52	0.119	163.37	–	0.85	4.25
	FLOWS	33.08	68.48	308.27	552.80	0.088	141.91	117.01	205.78	12.38
C33	CBS	39.58	48.79	428.27	766.55	0.202	299.89	–	56.71	3.84
	CPU	18.11	54.70	391.27	700.95	0.149	241.48	–	1.15	4.33
	FLOWS	33.00	62.70	343.28	616.11	0.117	177.42	146.41	160.66	11.22
C34	CBS	30.07	55.69	383.98	688.83	0.131	234.07	–	38.30	3.91
	CPU	16.53	62.99	340.95	612.92	0.117	175.58	–	1.56	3.44
	FLOWS	33.00	65.75	325.09	583.77	0.090	158.99	92.48	148.08	11.24
C35	CBS	22.69	62.28	345.48	620.42	0.097	182.61	–	75.51	5.97
	CPU	13.24	61.50	350.08	628.82	0.114	186.86	–	0.76	4.13
	FLOWS	32.05	58.02	371.17	664.62	0.141	212.22	119.30	157.69	12.90
C36	CBS	22.78	87.92	190.88	339.75	0.059	52.02	–	52.75	2.03
	CPU	10.75	87.51	193.21	343.62	0.048	54.83	–	0.20	2.60
	FLOWS	30.94	80.77	234.21	418.00	0.089	78.12	137.01	160.24	16.28
C37	CBS	24.27	80.98	232.25	414.96	0.051	80.90	–	30.99	3.20
	CPU	16.57	73.31	278.83	499.76	0.089	114.18	–	0.44	3.70
	FLOWS	33.08	80.94	233.02	415.99	0.050	81.28	57.60	178.18	11.11
C38	CBS	29.85	62.03	346.97	623.12	0.133	180.81	–	30.53	4.61
	CPU	10.63	60.12	357.66	642.19	0.169	192.15	–	0.66	3.19
	FLOWS	33.08	67.29	316.05	565.97	0.109	147.19	112.13	164.82	11.01
C39	CBS	28.29	62.92	341.04	613.38	0.100	177.63	–	43.73	3.86
	CPU	11.21	57.77	372.43	666.98	0.201	208.47	–	0.66	2.50
	FLOWS	33.00	67.67	313.59	562.04	0.070	148.53	115.58	158.22	8.81
C40	CBS	40.80	64.59	330.91	594.71	0.169	159.42	–	22.41‡	3.12
	CPU	18.80	61.94	347.14	623.81	0.111	183.56	–	0.78	3.99
	FLOWS	33.00	68.37	309.29	554.11	0.081	143.03	76.35	143.77	11.33
C41	CBS	35.07	86.76	197.61	351.61	0.070	55.41	–	19.17	3.01
	CPU	20.03	78.46	247.59	442.56	0.092	88.15	–	0.23	4.00
	FLOWS	33.00	81.24	231.24	412.75	0.069	78.15	70.90	163.68	8.08
C42	CBS	34.57	67.43	314.76	563.64	0.096	147.69	–	29.42	4.20
	CPU	20.48	65.81	323.59	581.41	0.118	156.06	–	0.73	4.79
	FLOWS	33.09	69.60	301.45	539.63	0.073	136.42	149.74	144.20	6.79
Mean	CBS	31.11	66.04	322.62	578.10	0.120	175.95	–	39.12†	3.26
	CPU	15.23	62.55	343.64	616.24	0.145	191.65	–	0.75	3.63
	FLOWS	32.28	68.77	306.63	548.96	0.091	147.82	111.47	168.44	11.16

G Additional Layout Comparisons

Figures 7–9 show 12 illustrative floor-qualified pairs and all seven below-floor pairs from the comparator-specific utilization-floor experiment (Section 4.3), rather than the fixed-threshold main experiment. The floor-qualified examples span different geometries and comparators and include throughput losses. Each pair uses a common coordinate system and scale; different warehouses have different scales. Labels marked "rot." indicate a joint 90° rotation for display. Empty space denotes unoccupied area, not necessarily a verified aisle.

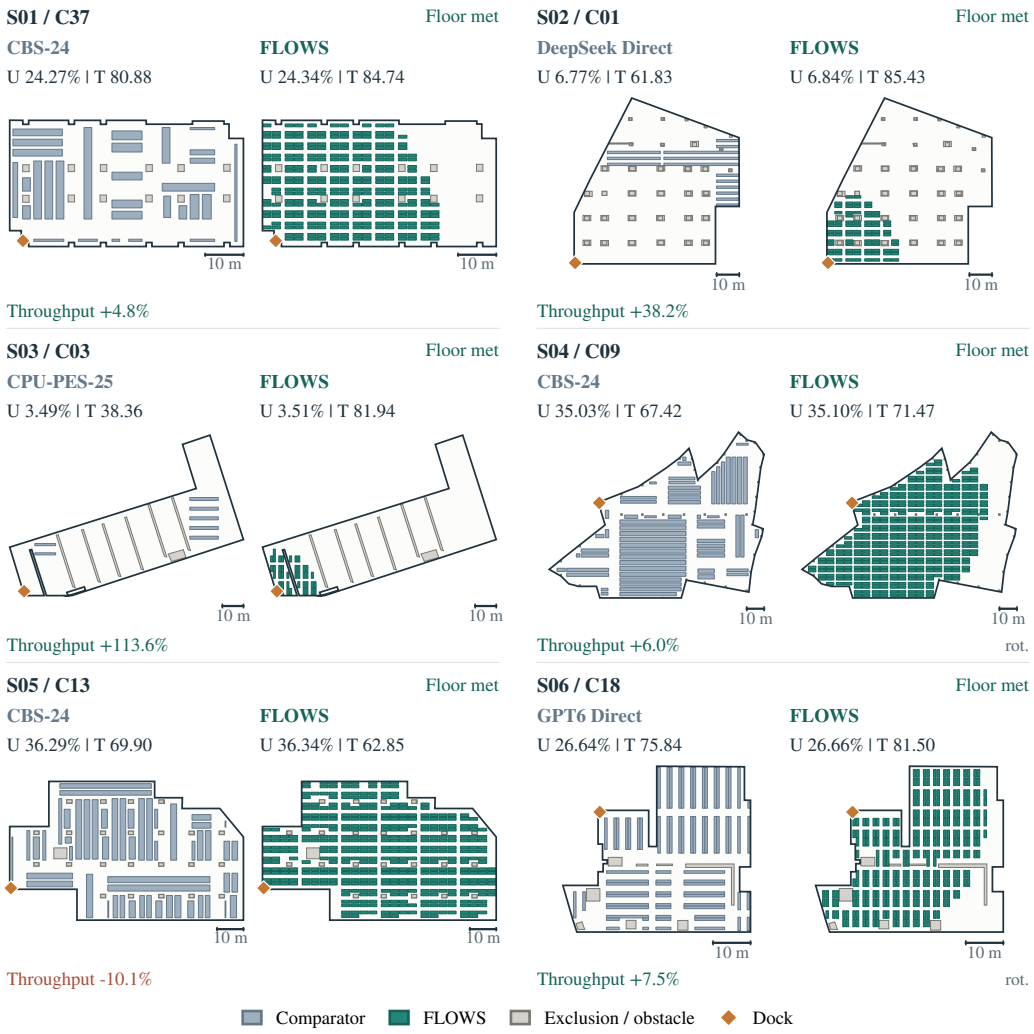

Figure 7: Floor-qualified comparator pairs S01–S06. Each comparator (left) is paired with the FLOWS layout generated under its footprint-utilization floor (right). U is footprint utilization (%) and T is throughput (orders/h), averaged over 20 confirmation replications. S05 illustrates that satisfying the utilization floor does not guarantee a throughput improvement.

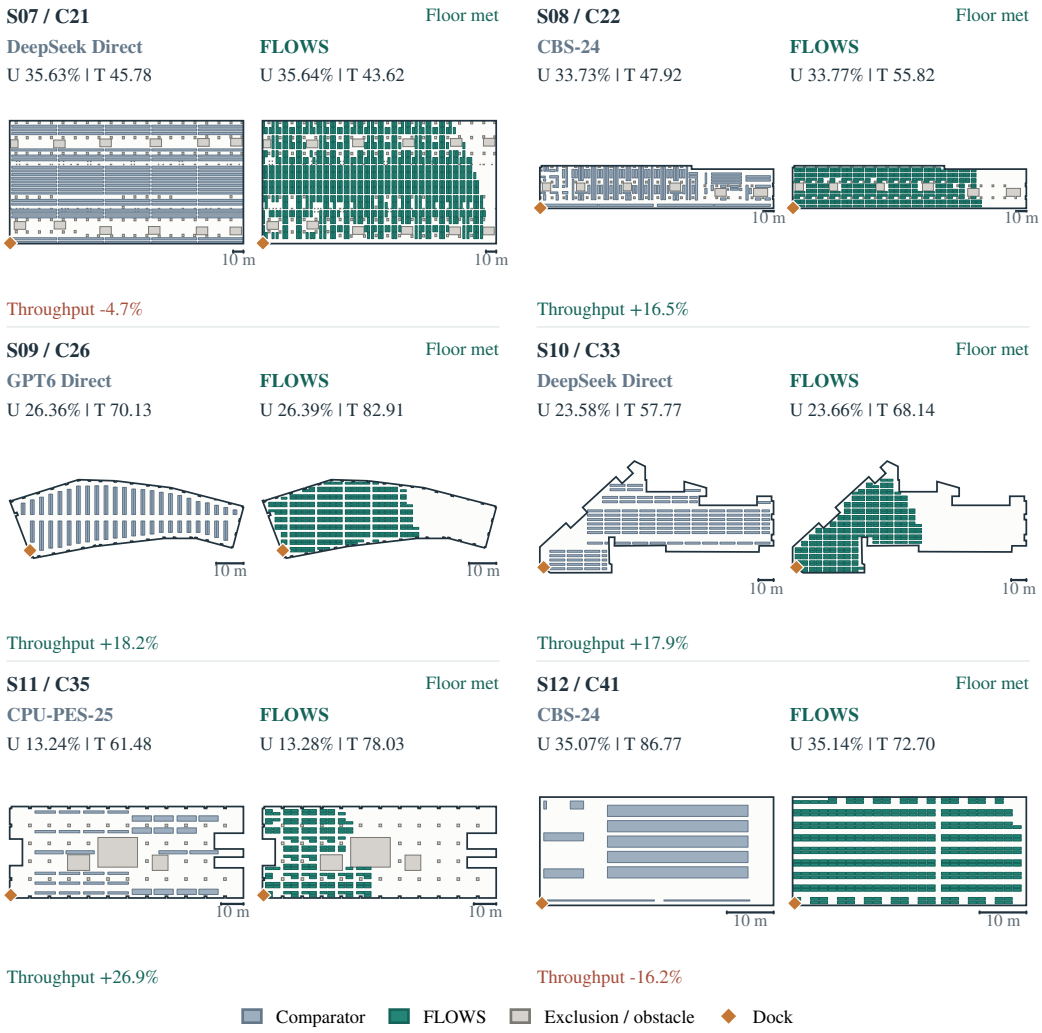

Figure 8: Floor-qualified comparator pairs S07–S12, using the same notation as Figure 7. These examples include long, narrow footprints and detailed boundaries; S07 and S12 retain throughput decreases. The selected examples illustrate layout variation; aggregate performance is reported over the full cohort.

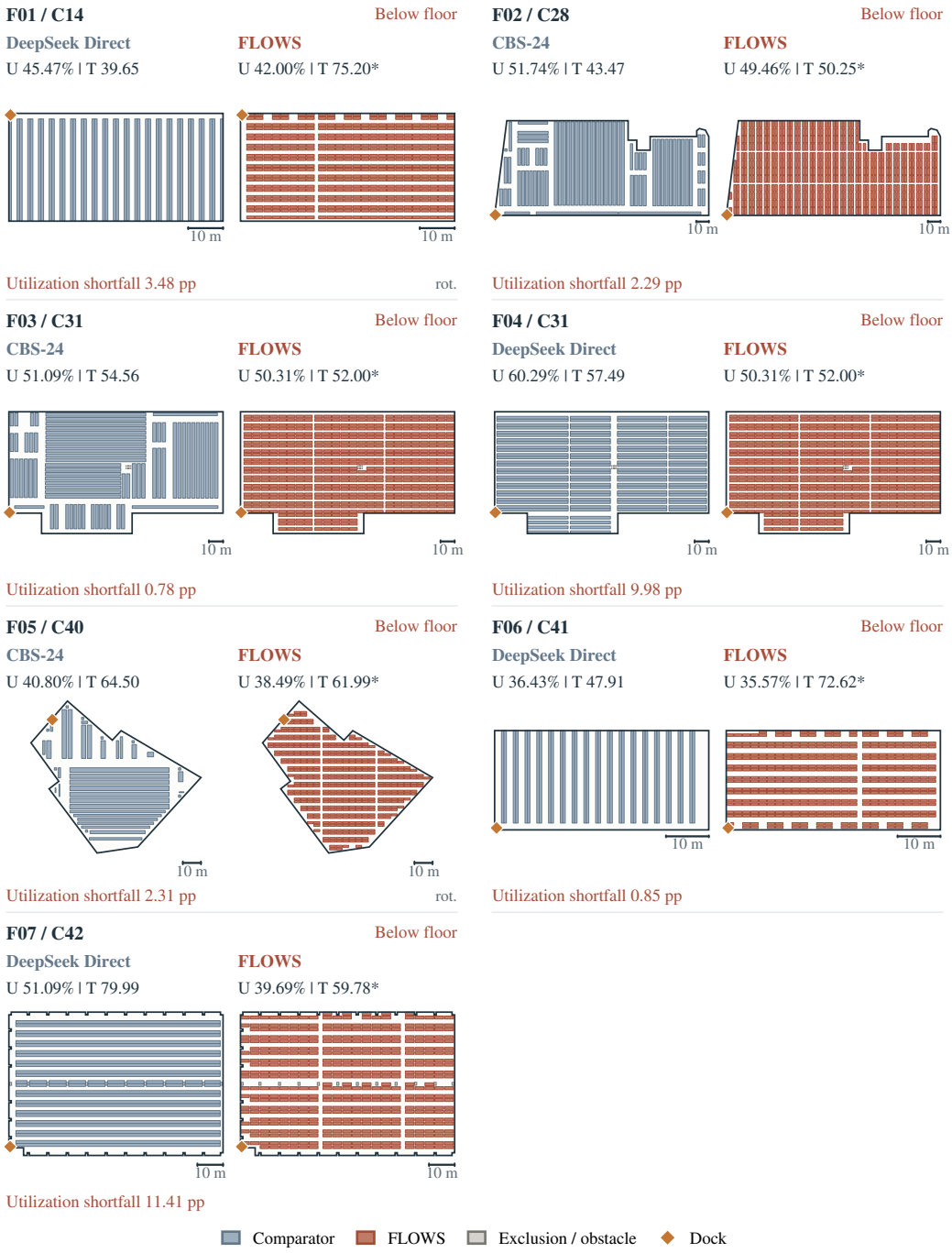

Figure 9: All seven below-floor pairs, covering six warehouses. Brown FLOWS layouts miss their comparator's utilization floor and remain failures. Starred throughput values come from separate diagnostic simulations of the unchanged layouts and are excluded from successful-pair summaries. C31 appears twice because CBS-24 and DeepSeek Direct impose different floors; pp denotes percentage points.