FACTORIO DESIGN BENCHMARK: EVALUATING LLM AGENTS ON HIERARCHICAL DESIGN SYNTHESIS

Anonymous authors
Paper under double-blind review

ABSTRACT

Designing a circuit to meet functional requirements or a factory to meet production targets exemplifies hierarchical design synthesis. These tasks require selecting component types and quantities, configuring and placing instances, organizing them into subsystems, and coordinating interfaces and shared infrastructure under coupled feasibility constraints and system-level objectives. Existing benchmarks evaluate important aspects of engineering design, but provide limited evidence on how well large language model (LLM) agents jointly coordinate these decisions across a complete system. We introduce Factorio Design Benchmark, which is based on Factorio, a deterministic factory-construction game. To complete the task, agents are required to submit complete declarative JSON factory designs for static validation, game execution, and production monitoring to decide whether it reaches the target production rate. We evaluate four LLMs on 32 tasks (8 target products $\times$ 2 product-specific target production rates $\times$ 2 map settings), with three independent runs per model–task pair. Across all runs, 73.7% produce the target product, but only 40.9% reach the target rate. Beyond these aggregate results, we examine how success varies with task demands. We derive a five-dimensional difficulty profile from task specifications, covering recipe complexity, capacity, logistics, spatial layout, and power infrastructure. The average of its five scores correlates negatively with the proportion of successful runs per task for each model, with Spearman $\rho = -0.90$ when runs are pooled across all four models. This association supports the profile as a proxy for relative task difficulty within the suite. To identify recurring design failures, we also analyze validation feedback. Entity overlap and unpowered consumers have the highest reported violation counts, with larger counts at high target production rates. These failures highlight challenges in coordinating spatial layouts and shared infrastructure across the complete factory.

1 INTRODUCTION

Many engineering design tasks require coordinated decisions at the component, subsystem, and system levels. In electronic design automation, designing a circuit to meet functional, timing, and power requirements involves selecting and configuring components, organizing them into functional modules, and coordinating their placement and connections (Huang et al., 2021). In production-system design, meeting a target production rate requires choosing equipment types and quantities, configuring and arranging machines into production stages, and coordinating material flow and power supply within a given site (Pérez-Gosende et al., 2021). In both settings, component choices determine subsystem behavior, while interfaces and shared resources couple decisions across the complete system. A change that improves one subsystem can increase resource demands or violate constraints elsewhere, so local design quality does not guarantee a feasible, high-performing system.

We refer to these tasks as *hierarchical design synthesis*. Given available component types and configuration options, interaction rules, feasibility constraints, and a system-level objective, the solver determines component types and quantities, configures and places component instances, and organizes them into subsystems with compatible interfaces. A complete design is represented as $D = (H, \{(t_j, \sigma_j)\}_{j=1}^{n})$, where n is the number of component instances, t_j and σ_j specify each instance's type and configuration, and H describes subsystem organization and interfaces. The goal is to satisfy all feasibility constraints while optimizing the system-level objective. A more detailed task formulation is provided in Appendix A.7.

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 (LLMs) are increasingly used for component design, design-flow automation, and broader design and manufacturing workflows (Fu et al., 2023; Wu et al., 2024; 2025; Makatura et al., 2024). These advances raise the question of whether agents can synthesize complete systems under coupled constraints. A locally plausible subsystem may become infeasible after integration, while capacity errors may become apparent only during execution. These cross-subsystem dependencies motivate evaluation on hierarchical design tasks.

Existing benchmarks assess functional artifact generation, individual design stages, design-flow automation, and factory construction (Liu et al., 2023; Lu et al., 2024; Yu et al., 2026; Wu et al., 2024; Hopkins et al., 2025). However, these settings often cover only part of the hierarchical design process or delegate key design decisions to automated tools. They therefore leave open whether LLM agents can coordinate decisions across design levels to produce complete systems that satisfy coupled constraints and system-level objectives.

We therefore introduce the *Factorio Design Benchmark*, which evaluates factory design in Factorio (Wube Software, 2020), a deterministic factory-construction game environment. The suite contains 32 tasks formed by crossing eight target products with two product-specific target production rates and two map settings. Each submission is a complete declarative JSON factory design. Evaluation combines deterministic static validation with game execution to check design constraints and measure target production and throughput. Experiments with four LLMs show that the suite is challenging, with failures in constructing valid designs, producing the target item, and meeting the target production rate. The benchmark provides a reusable evaluation environment and diagnostic tool for LLM agents on hierarchical design synthesis.

2 Related Work

2.1 LLMs for Engineering Design and Design Automation

LLMs are increasingly used to support engineering design. In hardware design, they have been used to generate hierarchical accelerator modules (Fu et al., 2023), explore analog circuit designs (Chen et al., 2024), and coordinate specialized agents for topology selection, sizing, and optimization (Shen et al., 2026). Execution feedback can also help correct local errors in hardware description language (HDL) code (Thakur et al., 2024). At the workflow level, ChatEDA (Wu et al., 2024) and EDAid (Wu et al., 2025) use task decomposition, script generation, and tool orchestration, while broader studies examine LLM support across design and manufacturing processes (Makatura et al., 2024). Nevertheless, performance declines as hardware designs grow larger and more hierarchical (Gomes Lopes et al., 2025), and numerical precision and multi-constraint reasoning remain important challenges (He & Yu, 2024; Pan et al., 2025). These findings motivate benchmarks that assess the capabilities and limitations of LLMs in engineering design, including their ability to carry out complete hierarchical design synthesis.

2.2 Benchmarks for Engineering Design Synthesis

Existing benchmarks cover several capabilities needed for hierarchical design synthesis. VerilogEval (Liu et al., 2023) evaluates generated Verilog through syntax and functional correctness, while RTLLM (Lu et al., 2024) further includes design-quality measures. ChipBench (Yu et al., 2026) evaluates Verilog generation, debugging, and reference-model generation. The placement benchmark ChiPBench (Wang et al., 2025) evaluates placement algorithms through their effects on final chip performance, power, and area. These benchmarks focus on circuit implementation and verification or individual design stages, leaving open whether LLMs can jointly select and configure components, organize subsystems, and satisfy constraints across a complete system.

Broader benchmarks extend evaluation to constrained product design and heterogeneous engineering tasks. BikeBench (Regenwetter et al., 2025) evaluates bicycle designs under multiple objectives and constraints within a fixed parametric representation. EngDesign (Guo et al., 2025) covers tasks ranging from parameter selection and component design to system-level design, but complete hierarchical design synthesis is not a uniform requirement across its tasks. Within Factorio, Patterson et al. use constraint programming to optimize individual production modules with prescribed dimensions and material inputs (Patterson et al., 2023). The Factorio Learning Environment (Hopkins

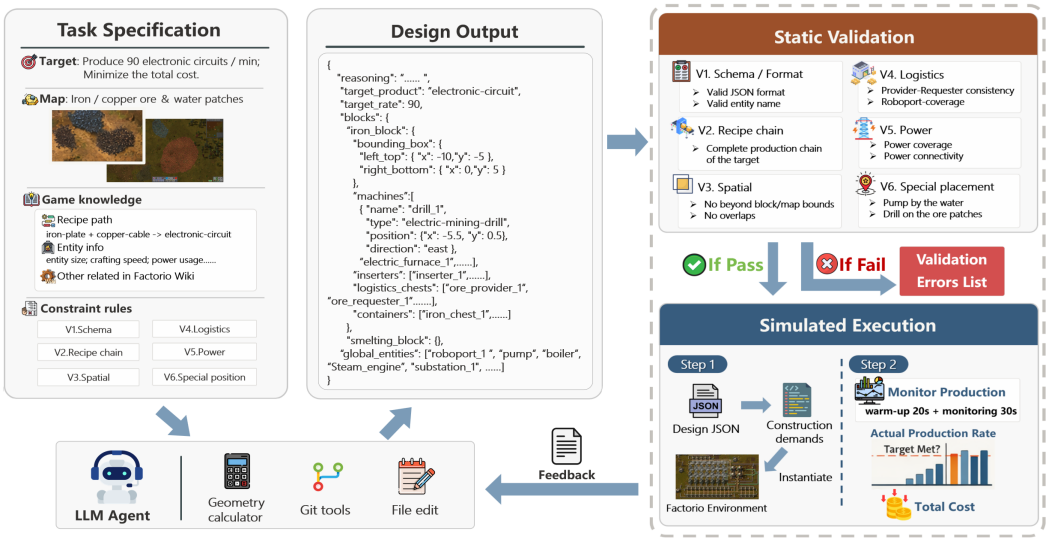


Figure 1: Overview of the benchmark pipeline. Given a task specification, an agent generates and submits a complete factory design in JSON. The evaluator applies deterministic static validation first, and designs that pass are instantiated in Factorio and monitored to measure target production and cost. Diagnostic feedback is returned to support subsequent revisions.

et al., 2025) supports interactive factory construction through incremental Python programs and environment feedback. Its high-level APIs automate some placement and routing decisions.

Prior work leaves limited controlled evidence about complete hierarchical design synthesis. Factorio Design Benchmark addresses this gap by requiring complete hierarchical design synthesis for every task in the suite. The task definition and suite construction are detailed in Section 3.1. The tasks exercise dependency, capacity, spatial, and interface reasoning together with system-level repair. Their relevance extends beyond Factorio at the level of task structure and required capabilities though direct transfer of scores remains an open question.

3 Factorio Design Benchmark

Factorio Design Benchmark instantiates hierarchical design synthesis as factory-design tasks in Factorio. The environment retains the coupled production, layout, and infrastructure decisions central to factory design (Pérez-Gosende et al., 2021), while its deterministic mechanics support reproducible evaluation. Figure 1 summarizes the task specification, design output, and evaluation pipeline, which are detailed in the following subsections.

3.1 Task Specification and Suite Construction

A task specifies a target product, a target production rate, map boundaries, and resource and water locations. Agents receive the production recipes, supported equipment and its properties, unit costs, operating rules, and design constraints. The agent must design a complete factory that satisfies these constraints and meets the target production rate. Among such designs, lower total entity cost is preferred. This cost is the sum of the unit costs of all placed entities and logistics robots. Unit costs are detailed in Appendix A.4. Figure 2 illustrates an example task and its operating mechanics.

Task suite. We construct the 32-task suite in Table 1 by combining eight target products with two target production rate levels (low and high) and two map settings (easy and hard). High-rate tasks require three to four times the throughput of their low-rate counterparts, increasing production, logistics, and power requirements. Hard maps combine smaller layout areas with changes to ore-patch sizes and locations, generally making equipment placement and infrastructure coordination more demanding. Complete task specifications are released as JSON, and are detailed in Appendix A.1.

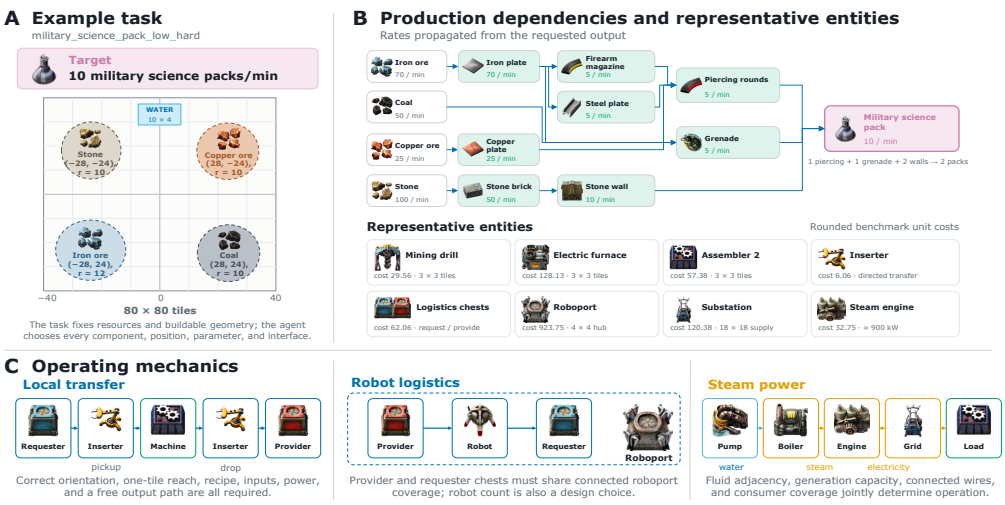


Figure 2: Task specification for `military_science_pack_low_hard`. **A** shows the production target and map. **B** presents production dependencies and representative equipment with unit costs. **C** illustrates local material transfer, robot logistics, and power supply. The required production rates of the intermediate products in **B** need to be inferred by the agent itself.

Table 1: The 32-task suite covers all combinations of eight target products, two product-specific target production rates (low/high), and two map settings (easy/hard). Production rates are in items per minute; map dimensions are in tiles and listed in easy/hard order.

Target product	Low	High	Map dimensions (easy/hard)
Iron plate	30	120	$50 \times 50 / 35 \times 35$
Iron gear	30	90	$50 \times 50 / 35 \times 35$
Electronic circuit	30	90	$70 \times 70 / 50 \times 50$
Automation science pack	30	90	$70 \times 70 / 50 \times 50$
Inserter (transfer arm)	20	60	$80 \times 80 / 60 \times 60$
Logistic science pack	15	45	$80 \times 80 / 60 \times 60$
Military science pack	10	30	$100 \times 100 / 80 \times 80$
Assembling machine 2	10	30	$100 \times 100 / 80 \times 80$

3.2 Specification-Derived Task Difficulty

Although the suite is constructed by varying product, target production rate, and map, these settings provide only a coarse characterization of task difficulty. Deeper or more highly branched recipe graphs require more production stages and material interfaces, which must fit within the available space and share logistics and power infrastructure. Coordinating these demands creates interdependent design decisions. For example, moving equipment to resolve a spatial conflict can disrupt material transfer or infrastructure coverage. To quantify these design demands, we derive a five-dimensional difficulty profile from task specifications, as shown in Table 2.

Table 2: Five dimensions of task difficulty and the features used to quantify them.

Dimension	Design demand	Features
Recipe complexity	Production depth, branching, and material diversity	Recipe-node count, dependency depth, extra prerequisite joins, raw-material type count.
Capacity demand	Equipment and raw-input requirements	Integer and fractional machine demand, maximum machines per stage, total raw-input rate.
Logistics/interface demand	Material-transfer interfaces and throughput	Non-fluid interface count and total/peak flow, transfer-channel count, machine input/output endpoints.
Spatial/terrain demand	Layout fit and resource access	Layout occupancy, maximum drill-to-ore capacity ratio, demand-weighted resource separation, water-obstacle fraction.
Power/shared-infrastructure demand	Generation, distribution, and water access	Combined power for machines and transfer, steam-engine count, substation coverage units, normalized water-access distance.

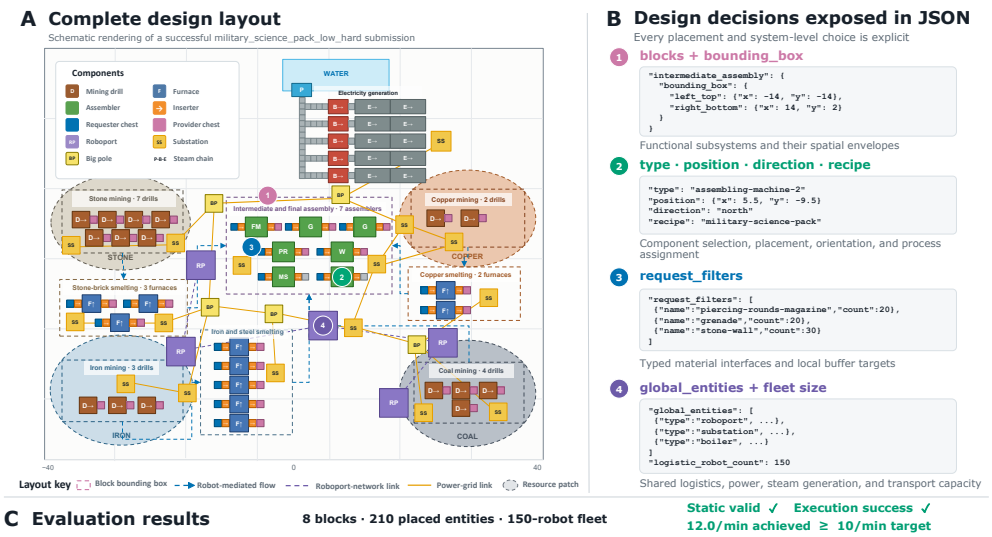


Figure 3: A complete factory design for `military_science_pack_low_hard`. **A** shows the schematic layout of the design. Matching numbered markers link the layout to the JSON excerpts in **B**, which illustrate block boundaries, entity configurations, material requests, and shared infrastructure. **C** reports the factory's scale and evaluation results.

We compute one difficulty score for each dimension from the corresponding features, yielding a five-dimensional profile for each task. The five scores can be analyzed separately or, when an ordering is needed, summarized by an overall difficulty index equal to 100 times their unweighted mean. The feature definitions, scoring procedure, and complete task profiles are provided in Appendix A.2.

3.3 Design Output

Each submission is a complete `ProductionLine` JSON object that organizes the factory into functional production blocks and their entities, as illustrated in Figure 3. Each block defines a bounding box and groups its machines, inserters, and chests. Entity fields specify component types, positions, directions, and, where applicable, recipes and requester filters. Provider and requester chests define material interfaces between entities or blocks, while top-level fields specify shared infrastructure and the logistics fleet. Appendix A.3 provides the full schema and a complete example.

Declarative JSON is well suited to the discrete components and nested structure of factory designs. It supports automated checking and translation into game-construction commands. Requiring design choices to be specified explicitly allows us to evaluate component selection, spatial layout, and interface coordination directly, with less confounding between design synthesis and code generation.

3.4 Validation and Execution

Evaluation proceeds in two stages. The first, static validation, checks each submitted design against 22 rules that specify necessary conditions for feasibility under the benchmark. The rules cover six constraint families, as summarized in Table 3, with representative failures illustrated in Figure 4. Only designs passing all rules proceed to the second stage.

The second stage attempts to instantiate the design in Factorio and, if successful, measures its target-item production rate after a warm-up period. We evaluate each submitted design using the following four metrics. Timing and production-rate measurement settings are provided in Appendix B.1. We allow a 5% tolerance for finite-window production variation when assessing target-rate success.

- **Static validity.** The design passes all 22 static rules.
- **Execution success.** The design is successfully instantiated in Factorio.
- **Target production.** The measured target-item production rate is positive.
- **Target-rate success.** The measured production rate reaches at least 95% of the target rate.

Table 3: Static validation rules in six constraint families. All rules must pass before game execution.

Category	Rule	Requirement
V1: Schema/Format	V1.1	Valid JSON structure and types matching the public schema
	V1.2	Every entity type belongs to the benchmark-supported subset
V2: Recipe Chain	V2.1	Recipe names are valid benchmark-supported recipes
	V2.2	The design contains producer machines for the complete target recipe chain
V3: Spatial	V3.1	Entity bounding boxes are contained within map bounds
	V3.2	Entity bounding boxes do not overlap
	V3.3	Entity centers follow footprint-dependent alignment rules
	V3.4	Block bounding boxes are valid, nonzero, and mutually disjoint
	V3.5	Every block-owned entity lies inside its block bounding box
V4: Logistics	V4.1	Logistics chests lie within a roboport coverage region
	V4.2	Requester chests declare valid positive request filters
	V4.3	Requester use is paired with provider-side supply
	V4.4	Multiple roboports form a connected logistics network
	V4.5	Inserter pickup and drop points reach valid source and target footprints
	V4.6	Robot logistics uses a positive top-level robot count
	V4.7	Production and storage entities have required local transfer endpoints
V5: Power	V5.1	Every electric consumer lies inside the supply area of connected distribution
	V5.2	The distribution graph connects consumers to a power source
V6: Terrain/Steam	V6.1	Mining drills are placed on compatible ore patches
	V6.2	Offshore pumps occupy valid water shoreline placements
	V6.3	Non-pump entities remain on land
	V6.4	Pump, boiler, engine, and pipe placements form a consistent fluid chain

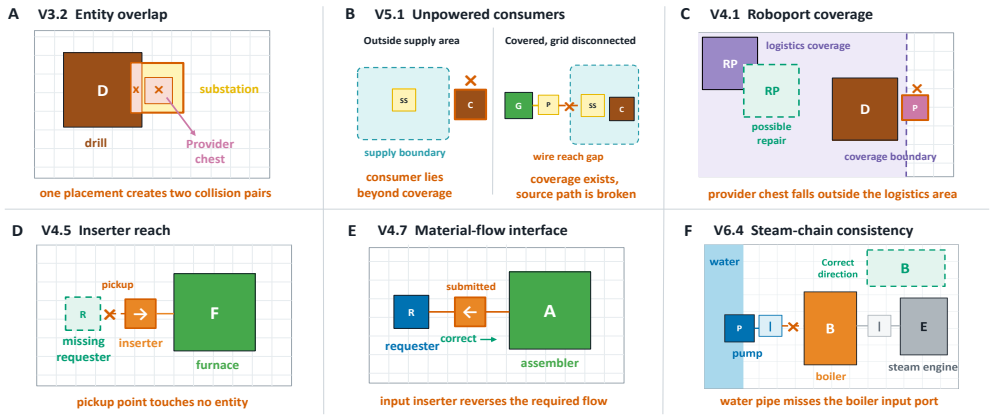


Figure 4: Six constructed examples of static-validation failures, with rule identifiers shown in each panel. Orange crosses mark failed relationships, and green annotations indicate possible local repairs. Appendix A.5 explains the individual panels.

4 Experiments

We use Factorio Design Benchmark to investigate three questions. First, how often do LLM agents succeed? Second, how does success vary across tasks, and where do they fail? Third, how does the cost of successful designs differ across models?

4.1 Experimental Setup

Agent. We evaluate DeepSeek V4 Flash and V4 Pro (April 24, 2026 preview release), GPT-5 mini, and Gemini 3 Flash Preview. Each model acts as a single agent that iteratively submits and revises complete factory designs using evaluator feedback. Agents have access to geometry utilities for distance and bounding-box calculations, file-editing tools, and Git for version control. The prompt template is provided in Appendix C.

Experimental protocol and metrics. We evaluate four models on the 32 tasks in Table 1, with three independent runs per model–task condition, yielding 384 completed runs. Each run allows at most 25 turns and 5 million aggregate model tokens, with at most one complete-design submission per turn. Model identifiers and further configuration details are provided in Appendix B.1.

We use the four metrics defined in Section 3.4 for evaluation. For each metric, a run is recorded as meeting the corresponding criterion if at least one of its submissions meets that criterion, regardless of later revisions. Our primary metric is run-level target-rate success, and a design or run meeting the target-rate criterion is termed successful. Model-level rates use all 96 runs for that model, while pooled task-level rates use the 12 runs for a task across the four models. Total entity cost is compared only among successful designs, as detailed in Section 4.4.

4.2 Overall Performance

Table 4: Run-level outcomes. The first four columns count runs with at least one qualifying submission, out of 96 per model (32 tasks with three runs each) or 384 pooled. The final column counts tasks with at least one target-rate-successful run out of three per model.

Model	Static validity	Execution success	Target production	Target-rate success	Tasks with ≥ 1 success
DeepSeek V4 Flash	77/96 (80.2%)	76/96 (79.2%)	74/96 (77.1%)	47/96 (49.0%)	19/32
DeepSeek V4 Pro	69/96 (71.9%)	69/96 (71.9%)	69/96 (71.9%)	45/96 (46.9%)	17/32
GPT-5 mini	53/96 (55.2%)	53/96 (55.2%)	49/96 (51.0%)	33/96 (34.4%)	12/32
Gemini 3 Flash Pre.	94/96 (97.9%)	94/96 (97.9%)	91/96 (94.8%)	32/96 (33.3%)	12/32
Pooled	293/384 (76.3%)	292/384 (76.0%)	283/384 (73.7%)	157/384 (40.9%)	20/32

Overall success remains limited, with the largest drop occurring between target production and target-rate success. Across all runs, 73.7% produce the target item, but only 40.9% attain target-rate success, as summarized in Table 4. The best model-level success rate is 49.0%, and 12 of 32 tasks remain unsolved across all four models.

The models differ in where runs fail. Most Gemini 3 Flash Preview runs attain static validity and target production, but many never meet the target-rate success criterion. Many GPT-5 mini runs never attain static validity, while the DeepSeek models exhibit both static-validation failures and throughput shortfalls. All 16 model–metric curves increase after turn 15, with turns 16–25 adding 8.3–9.1 percentage points to each pooled attainment rate (Appendix B.3). These late gains support the choice of a 25-turn budget for iterative refinement.

4.3 Cross-Task Variation and Failure Modes

To analyze cross-task variation in success, we relate task-level success rates to the overall difficulty index defined in Section 3.2. Across the 32 tasks, pooled target-rate success is strongly negatively correlated with the index (Spearman $\rho = -0.901$), as shown in Figure 5. The association holds for all four models ($\rho = -0.844$ to -0.776) and remains strong when dimension weights are varied or individual dimensions or product families are excluded (Appendix B.2).

All five dimensions are negatively associated with pooled task-level target-rate success. The correlations are strongest for power/shared-infrastructure demand ($\rho = -0.922$), capacity demand (-0.910), and logistics/interface demand (-0.884), followed by recipe complexity (-0.689) and spatial/terrain demand (-0.588). Capacity, logistics, and power demands are closely coupled through throughput requirements. Recipe complexity and spatial/terrain demand additionally characterize production dependencies and layout feasibility. These results support the profile as a useful proxy for relative task difficulty within this suite.

We next analyze reported rule violations to identify recurring design failures. The results point to spatial reasoning and coordination as the most prominent difficulties (Figure 6). Entity overlap (V3.2) and unpowered consumers (V5.1) have the highest reported multiplicities. Other recurring

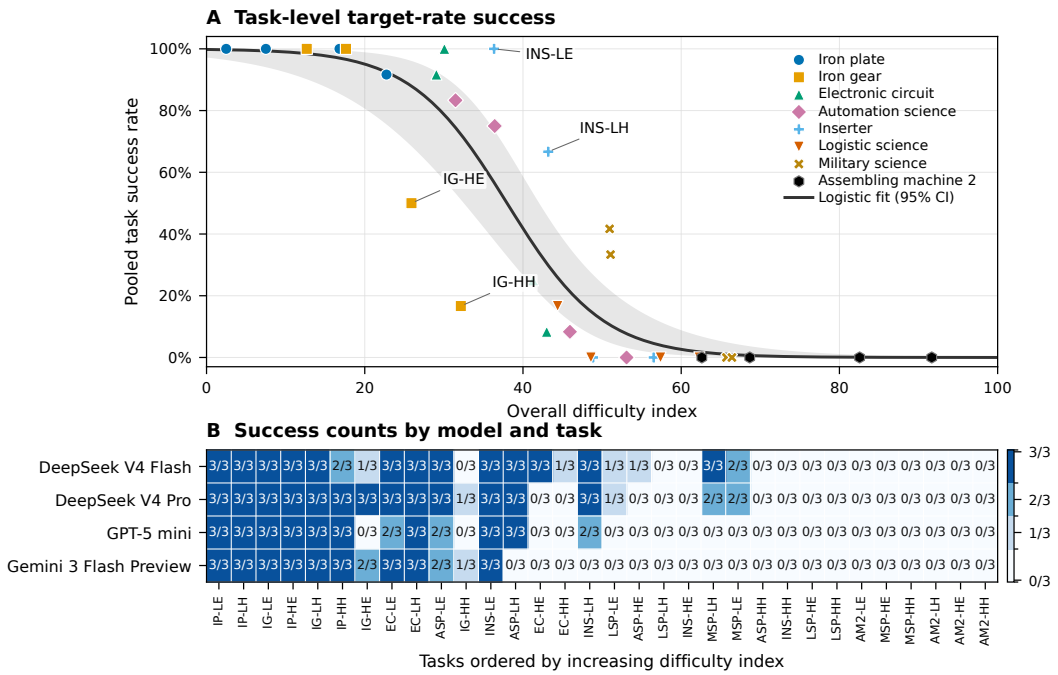


Figure 5: Target-rate success ordered by the overall difficulty index. **A** reports success fractions from 12 runs per task (four models with three runs each), with a logistic fit to all 384 binary run outcomes and a task-cluster-robust 95% confidence interval. **B** shows each model's task-level success count out of three runs, in the same task order.

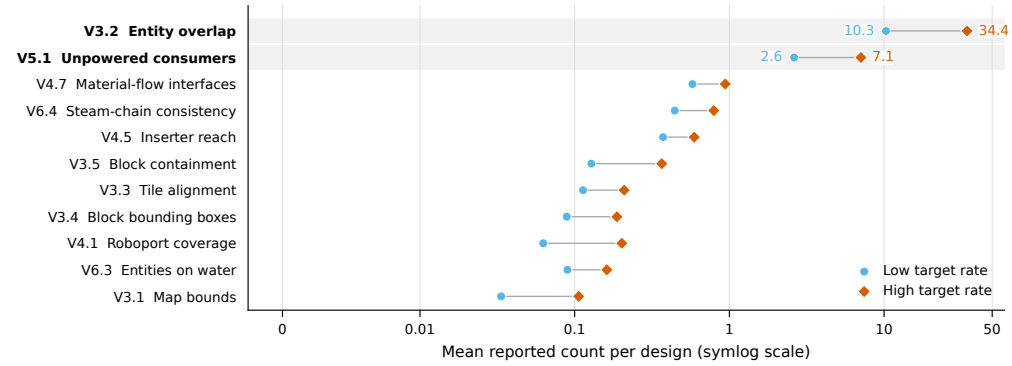


Figure 6: Mean reported violation counts at low and high target production rates across all 8,721 submissions. Absent rules contribute zero. Counts are averaged within each run, then equally over the 192 runs at each level. The 11 shown rules have a mean of at least 0.1 at either level. Some diagnostics cap or summarize counts. Counting details appear in Appendix B.2.

failures involve inserter reach (V4.5), material-flow interfaces (V4.7), roboport coverage (V4.1), and steam-chain directionality and adjacency (V6.4). Avoiding these failures requires coordinating component positions, orientations, and infrastructure connections across the complete factory.

Such coordination requires each placement to satisfy several constraints simultaneously. A consumer needs both local power coverage and a distribution grid connected to generation. Moving an entity to eliminate an overlap can disrupt transfer reach or power coverage, while adding poles introduces further placement constraints. Overlap and power failures frequently co-occur, with V3.2 present in 88.6% of submissions reporting V5.1. The larger reported violation counts at high target production rates are consistent with the greater coordination burden imposed by additional production equipment and supporting infrastructure. Figure 4 illustrates the relevant violations.

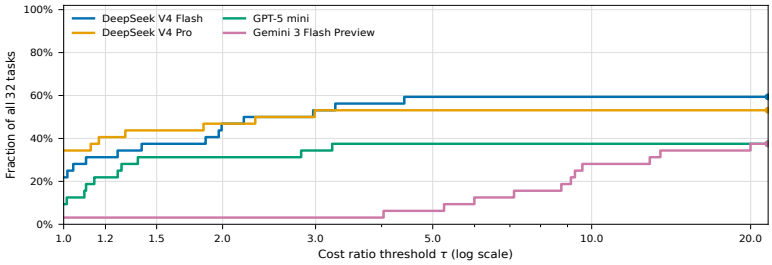


Figure 7: Cost performance profile. Each model's task cost is the minimum total entity cost over all target-rate-successful submissions across three runs. The profile reports the fraction of all 32 tasks within τ times the best observed successful cost for that task across the four models. Tasks without a successful submission from that model contribute zero but remain in the denominator.

4.4 Success-Aware Cost Performance

We next measure the resource efficiency of successful designs using total entity cost.

For model m, let $\mathcal{S}_m$ denote the tasks with at least one target-rate-successful submission across its three runs. For each $t \in \mathcal{S}_m$, let $C_{m,t}$ be the lowest total entity cost among all such submissions and $C_t^\star$ the best observed successful cost for task t across the four models. We define the success-aware cost performance profile as

$$P_m(\tau) = \frac{1}{32} \sum_{t \in \mathcal{S}_m} \mathbb{1}\left[\frac{C_{m,t}}{C_t^\star} \leq \tau\right], \qquad 1 \leq \tau < \infty.$$

Here, $\mathbb{1}[\cdot]$ is the indicator function. Tasks without a successful submission from model m contribute zero while remaining in the 32-task denominator, so the profile reflects both cost and coverage.

Figure 7 shows how cost competitiveness and task coverage yield different model rankings. At $\tau = 1$, Pro matches the best observed cost on 11/32 tasks, compared with 7/32 for Flash, while Flash achieves broader overall coverage. On the 16 tasks solved by both DeepSeek variants, Pro is cheaper on 11 tasks, Flash on four, and they tie on one. The geometric mean Flash/Pro cost ratio is 1.39. Pairwise comparisons on jointly solved tasks also show higher successful-design costs for Gemini. Higher success rates therefore do not necessarily imply more cost-efficient designs. Appendix B.2 provides detailed pairwise comparisons across all four models.

5 Conclusion

We introduced Factorio Design Benchmark, a 32-task benchmark for hierarchical design synthesis in Factorio. Agents specify complete factory designs in declarative JSON and receive feedback from static validation and game execution. Experiments with four LLMs reveal substantial challenges in meeting target production rates and coordinating spatial layouts and shared infrastructure. Higher success rates do not necessarily imply more cost-efficient designs. The overall index of our five-dimensional difficulty profile is negatively associated with success across tasks for every evaluated model, supporting the profile as a proxy for relative task difficulty within this suite.

Future work could equip agents with stronger tools for estimating equipment counts from production targets, detecting geometric conflicts, and diagnosing throughput bottlenecks. More informative feedback could suggest concrete repairs and identify which material interfaces or infrastructure connections a change may affect. Multi-agent systems could assign production planning, spatial layout, and infrastructure design to specialists coordinating through a shared design state and explicit interfaces. Reinforcement learning could use constraint satisfaction and measured throughput as verifiable rewards to train agents to construct and refine complete designs. Our evaluation prioritizes feasibility, target throughput, and total entity cost. We briefly discuss additional factory-design measures in Appendix A.6, with their applicability depending on the specific design requirements. Limitations of the benchmark and empirical evaluation are discussed in Appendix D.

AI USE STATEMENT

Generative AI tools assisted with software implementation, literature search, and manuscript drafting and revision. AI also provided feedback during author-led discussions of the experimental design. The research ideas, overall study direction, manuscript structure, and interpretation of results were determined by the authors. AI assistance with presenting the results focused on wording and clarity. The authors specified the content and presentation of figures, with AI assisting the implementation of plotting code. The authors reviewed AI-generated code and revised AI-assisted text, and take responsibility for the final implementation, manuscript, and findings.

ETHICS STATEMENT

This study uses synthetic factory-design tasks and model-generated designs in a simulated environment. It does not involve human participants or personal data.

REPRODUCIBILITY STATEMENT

Section 3 specifies the tasks, design representation, and evaluation procedure. Appendix A provides the detailed schema, difficulty-profile calculation, and entity costs. Section 4.1 describes the agent interface, run budgets, and metric aggregation, with model identifiers, evaluation dates, and execution settings documented in Appendix B.1. Appendix C provides the prompt template. An anonymized code repository is available at `https://anonymous.4open.science/r/Factorio-Design-Benchmark-6C35/`.

REFERENCES

Zihao Chen, Jiangli Huang, Yiting Liu, Fan Yang, Li Shang, Dian Zhou, and Xuan Zeng. Artisan: Automated operational amplifier design via domain-specific large language model. In *Proceedings of the 61st ACM/IEEE Design Automation Conference*, DAC '24, New York, NY, USA, 2024. Association for Computing Machinery. ISBN 9798400706011. doi: 10.1145/3649329.3655903. URL https://doi.org/10.1145/3649329.3655903.

Yonggan Fu, Yongan Zhang, Zhongzhi Yu, Sixu Li, Zhifan Ye, Chaojian Li, Cheng Wan, and Yingyan Celine Lin. Gpt4aigchip: Towards next-generation ai accelerator design automation via large language models. In *2023 IEEE/ACM International Conference on Computer Aided Design (ICCAD)*, pp. 1–9, 2023. doi: 10.1109/ICCAD57390.2023.10323953.

Samuel Gomes Lopes, Shien Zhu, and Gustavo Alonso. Exploring large language models for hierarchical hardware circuit and testbench generation. *ACM Trans. Des. Autom. Electron. Syst.*, 30 (6), October 2025. ISSN 1084-4309. doi: 10.1145/3742430. URL https://doi.org/10.1145/3742430.

Xingang Guo, Yaxin Li, XiangYi Kong, YILAN JIANG, Xiayu Zhao, Zhihua Gong, Yufan Zhang, Daixuan Li, Tianle Sang, Beixiao Zhu, Gregory Jun, Yingbing Huang, Yiqi Liu, Yuqi Xue, Rahul Dev Kundu, Qi Lim, Yizhou Zhao, Luke Granger, Mohamed Younis, Darioush Keivan, Nippun Sabharwal, Shreyanka Sinha, Prakhar Agarwal, Kojo Vandyck, Hanlin Mai, Zichen Wang, Aditya Venkatesh, Ayush Barik, Jiankun Yang, Chongying Yue, Jingjie He, Libin Wang, Licheng Xu, Hao Chen, Jinwen Wang, Liujun Xu, Rushabh Shetty, Ziheng Guo, Dahui Song, Manvi Jha, Weijie Liang, Weiman Yan, Bryan Zhang, Sahil Bhandary Karnoor, Jialiang Zhang, Rutva Pandya, Xinyi Gong, Mithesh Ganesh, Feize Shi, Ruiling Xu, Yifan Zhang, Yanfeng Ouyang, Lianhui Qin, Elyse Rosenbaum, Corey Snyder, Peter Seiler, Geir Dullerud, Xiaojia Zhang, Zuofu Cheng, Pavan Kumar Hanumolu, Jian Huang, Mayank Kulkarni, Mahdi Namazifar, Huan Zhang, and Bin Hu. Toward engineering agi: Benchmarking the engineering design capabilities of llms. In D. Belgrave, C. Zhang, H. Lin, R. Pascanu, P. Koniusz, M. Ghassemi, and N. Chen (eds.), *Advances in Neural Information Processing Systems*, volume 38, Main Conference. Curran Associates, Inc., 2025. doi: 10.52202/085713-2372. URL https://proceedings.neurips.cc/paper_files/paper/2025/file/664f777548205fb6e0cbb0965e8d2e16-Paper-Datasets_and_Benchmarks_Track.pdf.

Zhuolun He and Bei Yu. Large language models for eda: Future or mirage? In *Proceedings of the 2024 International Symposium on Physical Design*, ISPD '24, pp. 65–66, New York, NY, USA, 2024. Association for Computing Machinery. ISBN 9798400704178. doi: 10.1145/3626184.3639700. URL https://doi.org/10.1145/3626184.3639700.

Jack Hopkins, Mart Bakler, and Akbir Khan. Factorio learning environment. In D. Belgrave, C. Zhang, H. Lin, R. Pascanu, P. Koniusz, M. Ghassemi, and N. Chen (eds.), *Advances in Neural Information Processing Systems*, volume 38, Main Conference. Curran Associates, Inc., 2025. doi: 10.52202/085713-2745. URL https://proceedings.neurips.cc/paper_files/paper/2025/file/75f3a62e55187bc3e5e5961fa2a46bcf-Paper-Datasets_and_Benchmarks_Track.pdf.

Guyue Huang, Jingbo Hu, Yifan He, Jialong Liu, Mingyuan Ma, Zhaoyang Shen, Juejian Wu, Yuanfan Xu, Hengrui Zhang, Kai Zhong, Xuefei Ning, Yuzhe Ma, Haoyu Yang, Bei Yu, Huazhong Yang, and Yu Wang. Machine learning for electronic design automation: A survey. *ACM Trans. Des. Autom. Electron. Syst.*, 26(5), June 2021. ISSN 1084-4309. doi: 10.1145/3451179. URL https://doi.org/10.1145/3451179.

Mingjie Liu, Nathaniel Pinckney, Brucek Khailany, and Haoxing Ren. Invited paper: Verilogeval: Evaluating large language models for verilog code generation. In *2023 IEEE/ACM International Conference on Computer Aided Design (ICCAD)*, pp. 1–8, 2023. doi: 10.1109/ICCAD57390.2023.10323812.

Yao Lu, Shang Liu, Qijun Zhang, and Zhiyao Xie. Rtllm: An open-source benchmark for design rtl generation with large language model. In *2024 29th Asia and South Pacific Design Automation Conference (ASP-DAC)*, pp. 722–727, 2024. doi: 10.1109/ASP-DAC58780.2024.10473904.

Liane Makatura, Michael Foshey, Bohan Wang, Felix Hähnlein, Pingchuan Ma, Bolei Deng, Megan Tjandrasuwita, Andrew Spielberg, Crystal Elaine Owens, Peter Yichen Chen, Allan Zhao, Amy Zhu, Wil J. Norton, Edward Gu, Joshua Jacob, Yifei Li, Adriana Schulz, and Wojciech Matusik. Large Language Models for Design and Manufacturing. *An MIT Exploration of Generative AI*, mar 27 2024. URL https://doi.org/10.21428/e4baedd9.745b62fa. https://mit-genai.pubpub.org/pub/nmypmnhs.

Jingyu Pan, Guanglei Zhou, Chen-Chia Chang, Isaac Jacobson, Jiang Hu, and Yiran Chen. A survey of research in large language models for electronic design automation. *ACM Trans. Des. Autom. Electron. Syst.*, 30(3), February 2025. ISSN 1084-4309. doi: 10.1145/3715324. URL https://doi.org/10.1145/3715324.

Sean Patterson, Joan Espasa, Mun See Chang, and Ruth Hoffmann. Towards automatic design of factorio blueprints, 2023. URL https://arxiv.org/abs/2310.01505.

Pablo Pérez-Gosende, Josefa Mula, and Manuel Díaz-Madroñero. Facility layout planning. an extended literature review. *International Journal of Production Research*, 59(12):3777–3816, 2021. doi: 10.1080/00207543.2021.1897176. URL https://doi.org/10.1080/00207543.2021.1897176.

Lyle Regenwetter, Yazan Abu Obaideh, Fabien Chiotti, Ioanna Lykourentzou, and Faez Ahmed. Bikebench: A bicycle design benchmark for generative models with objectives and constraints. In D. Belgrave, C. Zhang, H. Lin, R. Pascanu, P. Koniusz, M. Ghassemi, and N. Chen (eds.), *Advances in Neural Information Processing Systems*, volume 38, Main Conference. Curran Associates, Inc., 2025. doi: 10.52202/085713-0678. URL https://proceedings.neurips.cc/paper_files/paper/2025/file/1d057f87f694734222a390ac3a373930-Paper-Datasets_and_Benchmarks_Track.pdf.

Jinyi Shen, Zihao Chen, Ji Zhuang, Jiangli Huang, Fan Yang, Li Shang, Zhaori Bi, Changhao Yan, Dian Zhou, and Xuan Zeng. Atelier: An automated analog circuit design framework via multiple large language model-based agents. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 45(1):31–44, 2026. doi: 10.1109/TCAD.2025.3573228.

Shailja Thakur, Jason Blocklove, Hammond Pearce, Benjamin Tan, Siddharth Garg, and Ramesh Karri. Autochip: Automating hdl generation using llm feedback, 2024. URL https://arxiv.org/abs/2311.04887.

Zhihai Wang, Zijie Geng, Zhaojie Tu, Jie Wang, Yuxi Qian, Zhexuan Xu, Ziyan Liu, Siyuan Xu, Zhentao Tang, Shixiong Kai, Mingxuan Yuan, Jianye Hao, Bin Li, and Feng Wu. Benchmarking end-to-end performance of ai-based chip placement algorithms. In D. Belgrave, C. Zhang, H. Lin, R. Pascanu, P. Koniusz, M. Ghassemi, and N. Chen (eds.), *Advances in Neural Information Processing Systems*, volume 38, Main Conference. Curran Associates, Inc., 2025. doi: 10.52202/085713-0673. URL https://proceedings.neurips.cc/paper_files/paper/2025/file/1cba8502063fab9df252a63968691768-Paper-Datasets_and_Benchmarks_Track.pdf.

Haoyuan Wu, Zhuolun He, Xinyun Zhang, Xufeng Yao, Su Zheng, Haisheng Zheng, and Bei Yu. Chateda: A large language model powered autonomous agent for eda. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 43(10):3184–3197, 2024. doi: 10.1109/TCAD.2024.3383347.

Haoyuan Wu, Haisheng Zheng, Zhuolun He, and Bei Yu. Divergent thoughts toward one goal: LLM-based multi-agent collaboration system for electronic design automation. In Luis Chiruzzo, Alan Ritter, and Lu Wang (eds.), *Proceedings of the 2025 Conference of the Nations of the Americas Chapter of the Association for Computational Linguistics: Human Language Technologies (Volume 1: Long Papers)*, pp. 1710–1721, Albuquerque, New Mexico, April 2025. Association for Computational Linguistics. ISBN 979-8-89176-189-6. doi: 10.18653/v1/2025.naacl-long.83. URL https://aclanthology.org/2025.naacl-long.83/.

Wube Software. Factorio. Video game, 2020. URL https://www.factorio.com/.

Zhongkai Yu, Chenyang Zhou, Yichen Lin, Hejia Zhang, Haotian Ye, Junxia Cui, Zaifeng Pan, Jishen Zhao, and Yufei Ding. Chipbench: A next-step benchmark for evaluating llm performance in ai-aided chip design, 2026. URL https://arxiv.org/abs/2601.21448.

A BENCHMARK SPECIFICATION

A.1 TASK NAMING AND MAP SPECIFICATIONS

The fixed 32-task suite, including product-specific target production rates and map dimensions, is summarized in Table 1. Each task name appends one of `low_easy`, `low_hard`, `high_easy`, and `high_hard` to its target product. These suffixes identify the target production rate and map settings used to construct the suite and do not enter the overall difficulty index. Easy and hard maps are paired within each product family and target production rate level. In addition to smaller map dimensions, each hard map changes resource locations, patch radii, and water rectangles. The released JSON specifications in `fd_benchmark/tasks/configs` provide the complete coordinates seen by the solver and used by the validator and task-difficulty calculator.

A.2 SPECIFICATION-DERIVED TASK DIFFICULTY DETAILS

The specification-derived task difficulty profile is versioned internally as `factorio_task_difficulty_v1`. It provides an optimistic, suite-relative operationalization of task-side design demand. Its values do not estimate feasible-region volume or observed design cost, and no model outcomes enter their computation. The calculator uses only task JSON, recipes, and pinned benchmark constants. Table 5 lists the raw features and transformations used to compute the five dimensions. For a recipe-graph node v, $\mathrm{indeg}(v)$ is the number of its immediate prerequisite inputs.

Table 5: Features used in the five difficulty dimensions. Count, load, flow, and power-magnitude features receive a $\log(1+x)$ transform before suite min–max normalization. The four spatial/terrain ratios and the normalized water-distance feature use the identity transform. Features are averaged equally within each dimension.

Dimension	Scored raw features	Transform
Recipe complexity	Recipe nodes (including mining); longest node depth including mining and target; $\sum_v \max(0, \mathrm{indeg}(v) - 1)$; raw-material type count	$\log(1+x)$
Capacity demand	Integer machine count, continuous machine load, largest stage, total raw-input rate	$\log(1+x)$
Logistics/interface demand	Non-fluid material interfaces (including final output), total and peak non-fluid interface flow, transfer-channel proxy, machine-level input/output endpoints	$\log(1+x)$
Spatial/terrain demand	Layout occupancy, maximum drill-to-ore capacity ratio, demand-weighted resource dispersion, water-obstacle fraction	identity
Power/shared-infrastructure demand	Estimated machine-plus-transfer active power, steam-engine count, distribution-coverage units, normalized anchor-to-water-boundary distance	first three $\log(1+x)$; distance identity

Starting from the target product and target production rate, we recursively expand the recipe chain toward its raw inputs to estimate production stages, machine loads, raw-material rates, and material-interface flows. Entity mechanics and the task's map, ore, and water geometry then yield the layout, resource-access, power-generation, and distribution features in Table 5.

Requirement propagation uses a fixed optimistic machine profile: assembling-machine-2 (speed 0.75, 150 kW, 3×3); electric furnace (speed 2.0, 180 kW, 3×3); and electric mining drill (speed 0.5, 90 kW, 3×3). Per-interface transfer demand uses an effective inserter rate of $0.86\times0.72=0.6192$ items/s and 15.1 kW per channel; its proxy is $\sum_e \lceil \mathrm{flow}_e/0.6192 \rceil$ over non-fluid material interfaces. Buildable area is map area minus the clipped union area of all water rectangles. Packed layout area divides the raw footprint of production machines, transfer channels, one final iron chest, generation entities, and fixed distribution entities by a density of 0.45. Power generation uses a 5% margin, 900 kW per steam engine, two engines per boiler, and an optimistic allowance of up to 400 engines per offshore pump; this pump allowance is a fixed sizing proxy rather than a claim about an optimal steam layout. Let A_{powered} denote the layout-area estimate used for electrical coverage, computed as the combined footprint of production machines, transfer entities, and steam engines divided by

the packing density of 0.45. Fixed-substation coverage uses a 324-tile^2 supply area at 0.65 coverage efficiency, with count $\lceil A_{\text{powered}}/(324 \times 0.65) \rceil$. Entity footprints, machine properties, and power constants follow the pinned task-demand calculator.

Some raw quantities are exact for the fixed profile (recipe closure, propagated rates, and resulting loads), while others are explicitly heuristic proxies. In particular, transfer-channel count is a throughput proxy; packed layout area assumes a common density; and distribution coverage assumes a fixed substation architecture. Ore capacity places non-overlapping 3×3 drills on a three-tile lattice and evaluates the nine phases in $\{0.5, 1.5, 2.5\}^2$, with validator-aligned 1.5-tile center margins, rather than solving a maximum independent-set problem. Candidate placements use the union of compatible resource patches and are clipped against map and water constraints. For multiple patches of one resource, demand is allocated in proportion to estimated capacity before demand-weighted pairwise separation is normalized by the map diagonal. Water access is the distance from the raw-material-demand-weighted resource center to the nearest water-rectangle boundary, also divided by the map diagonal. These proxies are held fixed across tasks and are never selected in response to model performance.

After transformation, every feature is min–max normalized over the fixed suite. Constant features would be assigned zero and excluded from the relevant mean; none is constant in the fixed suite. Within each dimension, the normalized features are combined with equal weight. The resulting five-dimensional vector is the primary profile and can be analyzed without further aggregation. For the single task ordering used in the main text, we additionally define the overall difficulty index as 100 times the equal-weight mean of the five scores. Let $D_t \in [0, 100]$ denote this index for task t. This aggregation is an explicit analysis choice. The profile itself remains the five-dimensional vector. Consequently, the normalized index and rank are suite-relative and may change when tasks are added. The five-dimensional vector retains more diagnostic information than the scalar index, while raw features are the more stable cross-suite quantities. The resulting task-level profiles are shown in Figure 8. Its overall column reports the unscaled mean $D_t/100 \in [0, 1]$, whereas the main text and ranking use $D_t \in [0, 100]$.

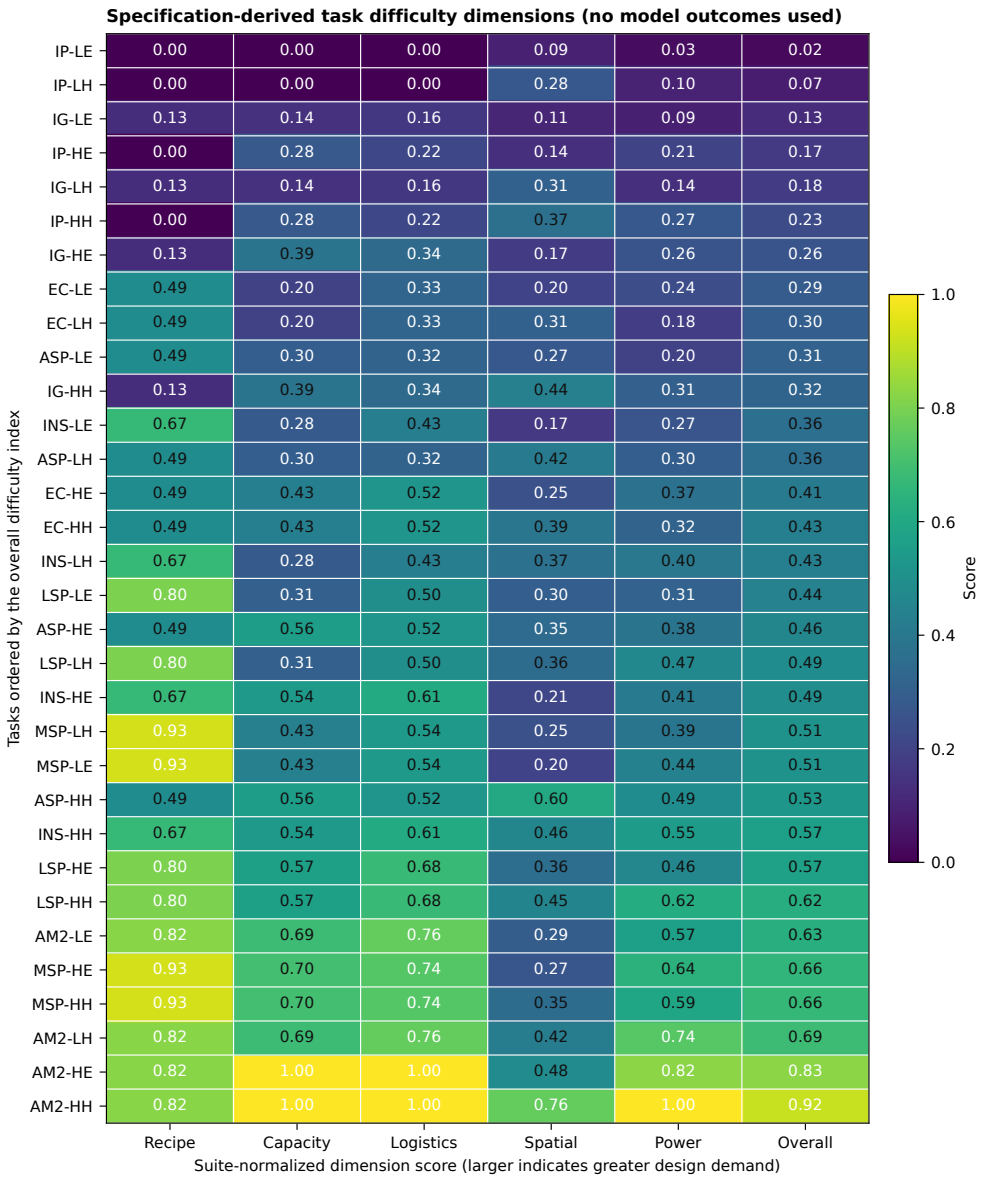


Figure 8: Specification-derived task difficulty profiles, ordered by the overall difficulty index. Each dimension aggregates raw features normalized within the fixed 32-task suite. The overall column reports the equal-weight summary used for aggregate analyses in the main text. No model outcomes are used.

A.3 Production Line Schema

Agents submit one `ProductionLine` JSON object. Table 6 summarizes its fields and nested object forms. All positions use global Factorio map coordinates, with x increasing to the right and y increasing downward. The optional `reasoning` and `summary` fields provide human-readable metadata and do not determine placement or execution. Fields with parser defaults may be omitted structurally, although the prompt asks agents to write the block collections explicitly. Semantic validation imposes additional conditions beyond schema parsing. Requester chests require positive request filters, and any design using logistics chests requires a positive `logistic_robot_count`. The executable schema and validator released with the benchmark remain authoritative.

Table 6: Condensed `ProductionLine` schema. Required fields have no parser default. Array and object defaults are shown explicitly.

Object	Field	Form	Purpose
`ProductionLine`	`reasoning`, `summary`	optional strings	Non-executable explanatory metadata
`ProductionLine`	`task_name`	required string	Task label repeated in the submission
`ProductionLine`	`target_product`	required string	Requested output item
`ProductionLine`	`target_rate_per_minute`	required number	Target production rate in items per minute
`ProductionLine`	`logistic_robot_count`	optional integer	Robots inserted into the logistics network, positive whenever logistics chests are used
`ProductionLine`	`blocks`	object of `Block`, default `{}`	Functional production blocks keyed by a block label
`ProductionLine`	`global_entities`	`GlobalEntity[]`, default `[]`	Shared logistics, electrical, water, and steam infrastructure
`Block`	`id`, `name`	required strings	Machine-readable identifier and human-readable name
`Block`	`description`	optional string	Human-readable block function
`Block`	`bounding_box`	required `BoundingBox`	Spatial region assigned to the block
`Block`	`machines`	`Machine[]`, default `[]`	Drills, furnaces, and assemblers
`Block`	`inserters`	`Inserter[]`, default `[]`	Local item-transfer entities
`Block`	`logistics_chests`	`LogisticsChest[]`, default `[]`	Provider and requester interfaces for robot logistics
`Block`	`containers`	`Container[]`, default `[]`	Regular storage containers, including final output
`Position`	`x`, `y`	required numbers	Global center coordinates for an entity or coordinates for a box corner
`BoundingBox`	`left_top`, `right_bottom`	two required `Positions`	Upper-left and lower-right corners of a block region
`Direction`	value	`north`, `south`, `east`, or `west`	Cardinal orientation encoded as a string
`Machine`	`name`, `type`, `position`	required fields	Entity identity, supported machine type, and center position
`Machine`	`direction`	`north` by default	Cardinal orientation
`Machine`	`recipe`	optional string	Assigned production recipe
`Inserter`	`name`, `position`, `direction`	required fields	Identity, center position, and drop direction
`Inserter`	`type`	string, default `inserter`	Supported inserter type
`LogisticsChest`	`name`, `type`, `position`	required fields	Identity, provider or requester type, and center position
`LogisticsChest`	`direction`	`north` by default	Cardinal orientation
`LogisticsChest`	`request_filters`	optional `RequestFilter[]`	Requested item buffers used by requester chests
`Container`	`name`, `position`	required fields	Identity and center position of regular storage
`Container`	`type`	string, default `iron-chest`	Supported regular-container type
`Container`	`direction`	`north` by default	Cardinal orientation
`GlobalEntity`	`name`, `type`, `position`	required fields	Identity, infrastructure type, and center position
`GlobalEntity`	`direction`	north by default	Cardinal orientation
`RequestFilter`	`index`, `name`, `count`	required integer, string, integer	Filter slot, requested item, and positive buffer count

A.3.1 Complete Successful Output

Figure 3 visualizes the larger military science pack factory used as the main-text example and selected fields from its JSON object. To show a complete serialization at readable length, Listing 1 reproduces a separate successful GPT-5 mini submission for `iron_plate_low_easy` from run 2, turn 5. The two blocks separate mining from smelting. A passive-provider and requester pair connects them through the shared roboport network, while the global entities provide steam generation and electrical distribution. The design passed all 22 static rules, placed all 15 listed entities, inserted both requested robots, and achieved 37.42 iron plates per minute against a target production rate of 30. The listing retains every submitted field and is intended as an illustrative output rather than a canonical or cost-optimal solution.

Listing 1: Complete `ProductionLine` output for `iron_plate_low_easy`, run 2, turn 5.


```
{
  "reasoning": "Single electric-mining-drill -> passive-provider chest -> requester chest ->
     electric-furnace. Logistics robots move ore from provider to requester. Steam power uses
     an offshore pump placed on the shoreline and a short land-side pipe feeding an east-
     facing boiler -> steam pipe -> east-facing steam engine. Fixed previous water-overlap by
     moving the offshore pump slightly north so only the pump touches the water and all pipes/
     boiler/engine are fully on land. Two robots provide low-cost logistics throughput and one
      substation covers generation while another substation covers the consumer cluster (
     furnace + roboport).",
  "task_name": "iron_plate_low_easy",
  "target_product": "iron-plate",
  "target_rate_per_minute": 30.0,
  "logistic_robot_count": 2,
  "blocks": {
    "mining_block": {
      "id": "mining_block",
      "name": "Iron Mining",
      "description": "Single drill mining iron ore and filling a passive-provider chest.",
      "bounding_box": {
        "left_top": {
          "x": -6,
          "y": -4
        },
        "right_bottom": {
          "x": 4,
          "y": 6
        }
      },
      "machines": [
        {
          "name": "drill_1",
          "type": "electric-mining-drill",
          "position": {
            "x": -0.5,
            "y": 0.5
          },
          "direction": "east"
        }
      ],
      "inserters": [],
      "logistics_chests": [
        {
          "name": "ore_provider",
          "type": "logistic-chest-passive-provider",
          "position": {
            "x": 1.5,
            "y": 0.5
          },
          "direction": "north"
        }
      ],
      "containers": []
    },
    "smelting_block": {
      "id": "smelting_block",
      "name": "Iron Smelting",
      "description": "Requests iron ore, smelts into iron plates, stores output in a chest.",
      "bounding_box": {
        "left_top": {
          "x": 6,
          "y": -4
        },
        "right_bottom": {
          "x": 16,
          "y": 6
        }
      },
      "machines": [
        {
          "name": "furnace_1",
          "type": "electric-furnace",
          "position": {
            "x": 10.5,
            "y": 0.5
          },
          "direction": "north",
          "recipe": "iron-plate"
        }
      ],
```

```
      "inserters": [
        {
          "name": "ins_furnace_in",
          "type": "inserter",
          "position": {
            "x": 8.5,
            "y": 0.5
          },
          "direction": "east"
        },
        {
          "name": "ins_furnace_out",
          "type": "inserter",
          "position": {
            "x": 12.5,
            "y": 0.5
          },
          "direction": "east"
        }
      ],
      "logistics_chests": [
        {
          "name": "ore_requester",
          "type": "logistic-chest-requester",
          "position": {
            "x": 7.5,
            "y": 0.5
          },
          "direction": "north",
          "request_filters": [
            {
              "index": 1,
              "name": "iron-ore",
              "count": 15
            }
          ]
        }
      ],
      "containers": [
        {
          "name": "plate_output_chest",
          "type": "iron-chest",
          "position": {
            "x": 13.5,
            "y": 0.5
          },
          "direction": "north"
        }
      ]
    }
  },
  "global_entities": [
    {
      "name": "roboport_1",
      "type": "roboport",
      "position": {
        "x": 4,
        "y": 0
      },
      "direction": "north"
    },
    {
      "name": "substation_consumers",
      "type": "substation",
      "position": {
        "x": 7,
        "y": -2
      },
      "direction": "north"
    },
    {
      "name": "substation_generation",
      "type": "substation",
      "position": {
        "x": 3,
        "y": -12
      },
      "direction": "north"
    },
    {
      "name": "pump_1",
```

```
      "type": "offshore-pump",
      "position": {
        "x": -0.5,
        "y": -17.5
      },
      "direction": "north"
    },
    {
      "name": "pipe_water_1",
      "type": "pipe",
      "position": {
        "x": -0.5,
        "y": -16.5
      },
      "direction": "north"
    },
    {
      "name": "boiler_1",
      "type": "boiler",
      "position": {
        "x": 0,
        "y": -14.5
      },
      "direction": "east"
    },
    {
      "name": "pipe_steam_1",
      "type": "pipe",
      "position": {
        "x": 1.5,
        "y": -14.5
      },
      "direction": "north"
    },
    {
      "name": "engine_1",
      "type": "steam-engine",
      "position": {
        "x": 4.5,
        "y": -14.5
      },
      "direction": "east"
    }
  ],
  "summary": "Minimal low-cost design: one electric-mining-drill mining iron (into a passive-
     provider chest), one requester chest feeding a single electric-furnace, inserters for
     input/output, two logistics robots and one roboport, and a small shoreline steam power
     cluster (offshore pump -> short land pipe -> east-facing boiler -> steam pipe -> steam
     engine). Pump placement revised so only the pump touches water and all pipes/boiler/
     engine are fully on land to satisfy water-overlap validation."
}
```


A.4 Entity Cost Table

We define a recipe-based unit-cost model, `factorio_recipe_cost_v1`, using the material requirements and crafting times of entity recipe chains. Fixed benchmark-specific weights convert these quantities into the scalar costs reported in Table 7. The table lists the 14 placeable entity types admitted by the public validation rules together with the per-robot cost. Every placed entity contributes its unit cost, and `logistic_robot_count` contributes the per-robot cost. Cost is not a validity constraint and is compared only among successful designs.

Table 7: Unit costs for the 14 supported placeable entity types and logistics robots.

Entity	Unit cost
assembling-machine-2	57.375
big-electric-pole	31.875
boiler	9.75
electric-furnace	128.125
electric-mining-drill	29.5625
inserter	6.0625
iron-chest	8.125
logistic-chest-passive-provider	62.0625
logistic-chest-requester	62.0625
logistic-robot	160.6875
offshore-pump	7.75
pipe	1.125
roboport	923.75
steam-engine	32.75
substation	120.375

Figure assets. Factorio item and entity icons in Figure 2 are © Wube Software and were sourced from the official Factorio Wiki.

A.5 Illustrative Validation Failures

Figure 4 illustrates six constructed failure cases. In Panel **A**, one provider chest overlaps both a mining drill and a substation. Panel **B** distinguishes a consumer outside a substation's supply area from a covered consumer whose distribution grid is disconnected from generation. Its symbols G, P, SS, and C denote a generator, power pole, substation, and electric consumer, respectively. Panel **C** places a provider chest outside roboport coverage.

Panel **D** shows an inserter whose pickup point reaches no source entity. Panel **E** reverses the direction required for material input, and Panel **F** misaligns the water connection with the boiler input. Shaded regions in Panels **B** and **C** indicate electrical supply and logistics coverage, respectively. Orange crosses mark failed relationships, while green annotations show one possible local repair where applicable.

A.6 Additional Design-Quality Measures

Feasibility, target throughput, and total entity cost correspond to the objectives specified by the current tasks. Additional measures could distinguish successful designs along other dimensions, provided that their definitions reflect the design question and operating conditions.

Layout efficiency can be assessed through occupied footprint or flow-weighted transport distance. These measures capture different aspects of a layout, since a compact arrangement need not minimize transport between heavily connected production stages. Resource utilization also requires a distinction between deployed equipment and runtime activity. Entity cost accounts for equipment expenditure, whereas machine utilization would require execution telemetry. Higher utilization can indicate less idle capacity but also less capacity margin for changes in demand, so its desirability depends on the operating objective.

Modularity could be examined through subsystem separation, interface consistency, and coupling between blocks. These properties differ from template reuse. The current output lists concrete entity instances, so measuring reuse directly would require an extension with parameterized block templates and explicit instantiation. Such an extension would make reusable abstractions part of the design output.

Robustness requires a specified perturbation model. Successful designs could be reevaluated under changes in demand, transport capacity, or power availability, or under component failures, with robustness measured by retained feasibility and throughput. The perturbations and their severity

would determine what the resulting scores mean. These measures therefore offer complementary evaluation directions for different design objectives.

A.7 Formal Task Definition

A hierarchical design synthesis task is specified by $(C, \Sigma, \mathcal{H}, R, f, \Phi)$. Here, C is the set of component types, $\Sigma = \{\Sigma_c\}_{c \in C}$ gives the configuration space for each type, and $\mathcal{H}$ contains the admissible subsystem hierarchies. The rules R define component dependencies, subsystem interfaces, and composition or execution semantics. Together, C, Σ, $\mathcal{H}$, and R induce the candidate design space $\mathcal{D}$.

A design $D = (H, \{(t_j, \sigma_j)\}_{j=1}^{n})$ selects n component instances with types $t_j \in C$ and configurations $\sigma_j \in \Sigma_{t_j}$, while $H \in \mathcal{H}$ assigns the instances to subsystems and records their interfaces. The constraint set $\Phi = \{\phi_1, \ldots, \phi_k\}$ defines the feasible design space $\mathcal{D}_{\text{feas}} = \{D \in \mathcal{D} \mid \phi(D) = \text{true for every } \phi \in \Phi\}$. The objective $f \colon \mathcal{D} \to \mathbb{R}$ ranks design quality, and the goal is to find an optimizing design $D^* \in \mathcal{D}_{\text{feas}}$.

Each engineering domain supplies its own components, representation, constraints, and execution model within this shared structure. For example, components may be machines, subsystems may be production stages, constraints may govern placement and connectivity, and the objective may measure deployment cost.

B Additional Experimental Details

B.1 Experiment Configuration

Table 8 records the requested model identifiers, evaluation dates, run budgets, and execution settings used for the main experiment.

Table 8: Main experiment configuration. Dates span the recorded evaluator timestamps for each model. Model status is reported as of evaluation, and version provenance is documented below.

Field	Value
Evaluation dates (2026)	DeepSeek V4 Flash, July 4–16; DeepSeek V4 Pro, July 8–16; GPT-5 mini, July 13–14; Gemini 3 Flash Preview, July 13–14
Model endpoints	DeepSeek V4 Flash Preview (official DeepSeek API, `deepseek-v4-flash`); DeepSeek V4 Pro Preview (official DeepSeek API, `deepseek-v4-pro`); GPT-5 mini (released model through OpenRouter, `openai/gpt-5-mini`); Gemini 3 Flash Preview (preview model through OpenRouter, `google/gemini-3-flash-preview`)
Solver	Single-agent iterative evaluation for every run
Tasks	All 32 tasks in Table 1
Independent runs	3 per model–task condition
Evaluation matrix	4 models $\times$ 32 tasks $\times$ 3 runs = 384 completed runs
Turn, call, and token budget	25 turns with at most one complete-design submission per turn; up to 20 model responses per response loop; at most 5 million aggregate model tokens per run
Factorio version	1.1.110
Execution timing	200 simulated seconds of warm-up and approximately 300 simulated seconds of monitoring, implemented as 20 and 30 wall-clock seconds at 10$\times$ nominal game speed. Production rates are computed from actual elapsed game ticks.
Parallel execution	Per-model concurrency limits of 12 or 24 across batches, with one active Factorio execution per server instance
Reported metrics	Static validity; execution success; target production; target-rate success; specification-derived task difficulty profile; success-aware cost performance profile

The warm-up period allows materials to propagate through the production chain and reduces initialization transients. The monitoring window averages production over multiple machine cycles to reduce sensitivity to bursty output.

Model-version provenance. According to the official release history, the DeepSeek endpoints used in our July evaluation correspond to the preview release introduced on April 24, 2026. The subsequent Flash and Pro updates were released on July 31 and August 13, respectively, under the same API identifiers.[1] Google introduced `gemini-3-flash-preview` on December 17, 2025 and lists this exact model code without a separate dated snapshot identifier.[2] OpenAI documents the GPT-5 mini snapshot `gpt-5-mini-2025-08-07`,[3] but our requests used OpenRouter's undated `openai/gpt-5-mini` identifier. Response-level model versions and system fingerprints were not retained in the experiment logs, which limits retrospective identification to the requested endpoints, evaluation dates, and documented release history.

[1] DeepSeek release announcement at `https://api-docs.deepseek.com/news/news260424/` and update history at `https://api-docs.deepseek.com/updates/`.

[2] Google release history at `https://ai.google.dev/gemini-api/docs/changelog` and model documentation at `https://ai.google.dev/gemini-api/docs/models/gemini-3-flash-preview`.

[3] `https://developers.openai.com/api/docs/models/gpt-5-mini`.

B.2 Additional Results

Difficulty associations and sensitivity. A 10,000-sample bootstrap over product families gives a 95% interval of $[-0.953, -0.788]$ for the pooled task-level Spearman correlation of -0.901. A run-level logistic regression with model fixed effects and task-clustered robust standard errors gives an odds ratio of 0.163 per ten-point increase in the index (95% confidence interval $[0.080, 0.333]$, $p < 10^{-6}$). Table 9 shows a monotonic decline in target-rate success across difficulty-index quartiles, with no successes in the highest quartile.

Table 9: Nested run-level outcome attainment across quartiles of the fixed overall difficulty index. Each entry counts runs that attain the indicated outcome at least once within the full run budget. Every quartile contains eight tasks and pools four models with three runs per model–task condition, giving 96 runs per quartile.

Difficulty-index quartile	Static validity	Execution success	Target production	Target-rate success
Q1 (lowest)	96/96	95/96	95/96	88/96
Q2	91/96	91/96	90/96	57/96
Q3	67/96	67/96	63/96	12/96
Q4 (highest)	39/96	39/96	35/96	0/96

Marginal task-level correlations with target-rate success are -0.922 for power/shared-infrastructure demand, -0.910 for capacity demand, -0.884 for logistics/interface demand, -0.689 for recipe complexity, and -0.588 for spatial/terrain demand. Capacity, logistics, and power are mechanically coupled and strongly correlated with one another (0.929–0.953). Recipe complexity is more strongly associated with static validity ($\rho = -0.752$) than with target-rate success.

The overall association remains strong when any one dimension is removed (ρ from -0.940 to -0.878) or any one product family is omitted (-0.922 to -0.871). Across 381 weighting schemes assigning each dimension 10% to 30% of the total weight, ρ remains between -0.929 and -0.868. Table 10 reports model-specific task-level correlations and leave-one-dimension-out checks. These analyses use the difficulty-profile definition unchanged, and no weight is fitted to success.

Controlled target production rate and map contrasts. Figure 9 reports target-rate success across the controlled target production rate and map variants. Increasing the target production rate raises all four demand dimensions other than recipe complexity in each of the 16 matched pairs, while pooled target-rate success falls from 63.0% to 18.8%. Hard maps increase spatial/terrain demand in all 16 matched pairs while leaving recipe complexity, capacity, and logistics/interface demand unchanged. Their success difference is smaller, from 44.3% on easy maps to 37.5% on hard maps. Each target production rate level or map setting pools 192 runs. These contrasts help explain the stronger marginal associations of the dimensions tied to throughput.

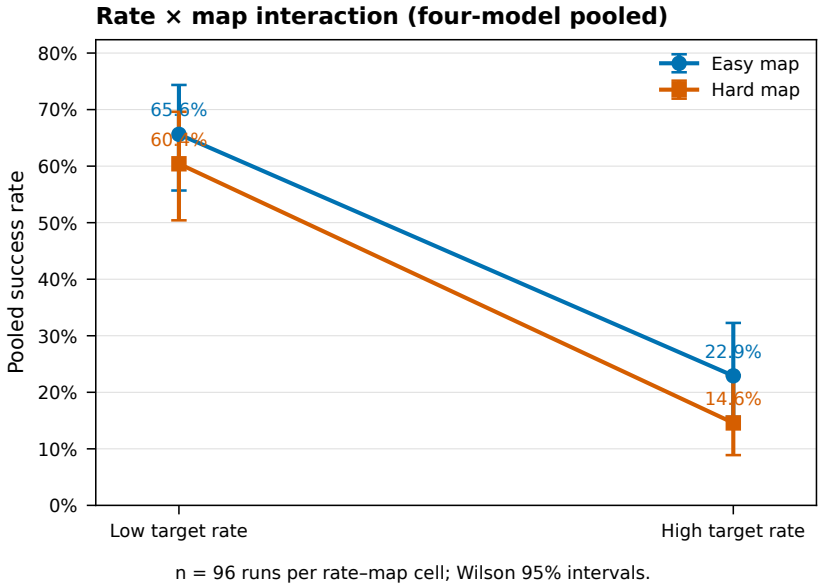


Figure 9: Pooled run-level target-rate success for low/high target production rates and easy/hard maps. Each combination of these settings contains 96 runs, comprising eight products, four models, and three runs per model–task condition. Each plotted success rate is the number of successful runs divided by 96, and bars show Wilson 95% confidence intervals.

Table 10: Robustness of the association between the overall difficulty index and task-level success. Model rows correlate the number of target-rate-successful runs out of three for each model–task condition with the index. Ablation rows recompute the index after removing one dimension and correlate it with the pooled task-level target-rate success rate, defined as successful runs out of 12 for each task.

Analysis	**Spearman ρ**
DeepSeek V4 Flash	−0.776
DeepSeek V4 Pro	−0.833
GPT-5 mini	−0.777
Gemini 3 Flash Preview	−0.844
Without Recipe	−0.940
Without Capacity	−0.878
Without Logistics	−0.899
Without Spatial	−0.880
Without Power	−0.893

Reported rule-violation counts. Figure 6 uses all 8,721 submissions. An absent rule contributes zero, and counts are first averaged over submissions within each run and then equally over the 192 runs at each target production rate level. The figure shows the 11 rules whose mean reaches 0.1 for at least one level. Some diagnostics cap or summarize examples, so cross-rule comparisons reflect reported counts rather than exhaustive violation totals. V3.2 records exact overlapping-entity pairs. V5.1 records unpowered consumers, except that a design with no electric pole is logged as one violation. The run-equal mean rises from 10.3 to 34.4 overlap pairs and from 2.6 to 7.1 reported V5.1 violations per design between low and high target production rates. Among target-rate-unsuccessful submissions affected by V5.1, 88.6% also contain entity overlaps.

Pairwise cost comparisons. Table 11 complements the success-aware cost performance profile with pairwise comparisons restricted to tasks solved by both models.

Table 11: Entity-cost comparison on common successful support. For each model–task condition, cost is the least-cost target-rate-successful design found across all submissions in its three runs. Common support contains tasks solved by both models. The ratio is A/B; intervals are task-bootstrap 95% confidence intervals. Win/tie/loss is from A's perspective.

A	B	Common	Geom. mean A/B	Median A/B	W/T/L
DeepSeek V4 Flash	DeepSeek V4 Pro	16	1.390 [1.027, 1.857]	1.254	4/1/11
DeepSeek V4 Flash	GPT-5 mini	12	1.190 [0.926, 1.627]	1.001	4/2/6
DeepSeek V4 Flash	Gemini 3 Flash Preview	11	0.178 [0.129, 0.250]	0.170	11/0/0
DeepSeek V4 Pro	GPT-5 mini	12	0.848 [0.639, 1.094]	0.947	7/1/4
DeepSeek V4 Pro	Gemini 3 Flash Preview	12	0.169 [0.107, 0.315]	0.141	11/0/1
GPT-5 mini	Gemini 3 Flash Preview	10	0.129 [0.093, 0.188]	0.112	10/0/0

B.3 Progress Across the Turn Budget

Figure 10 traces the accumulation of the four run-level outcomes. Across all 384 runs, turns 16–25 add 32–35 qualifying runs per metric. Gemini 3 Flash Preview illustrates how progress differs across outcomes. By turn 15, it has attained static validity and execution success in 91 runs and target production in 86, yet its target-rate successes rise from 21 to 32 over the final ten turns. Target-rate attainment therefore continues to increase after earlier outcomes approach their final levels.

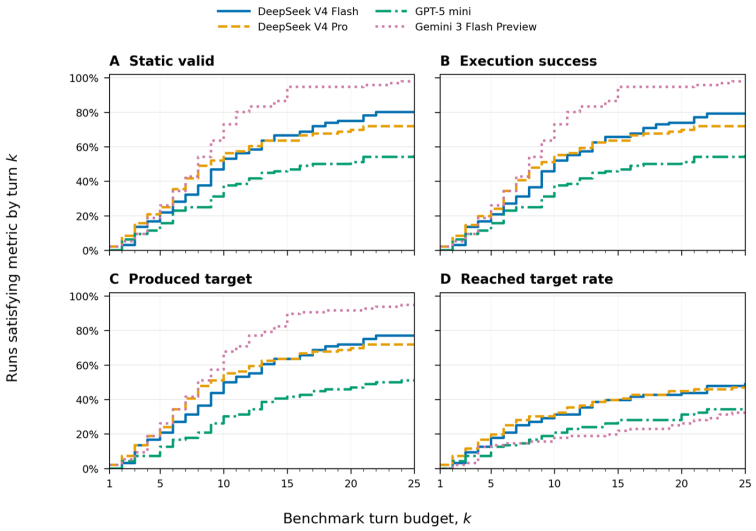


Figure 10: Cumulative run-level outcomes within the 25-turn and 5-million-token budgets. Each panel shows one metric across four models. For metric s and turn budget k, $s@k$ is the fraction of a model's 96 runs (32 tasks with three runs each) with at least one submission satisfying s by turn k. Once counted, a run remains counted at later turns regardless of subsequent revisions. The endpoints at $k = 25$ match the first four columns of Table 4.

C PROMPT TEMPLATES

This appendix presents a condensed version of the main prompt template used in the single-agent iterative evaluation. The executable prompt files are released under `fd_benchmark/prompts`. Shared task context, game-knowledge modules, and validation-rule text retain the template placeholders used in the implementation. The long in-context JSON example is omitted for brevity.

C.1 SINGLE-AGENT ITERATIVE PROMPT

System Prompt Template

```
You are a production line designer for Factorio.

{{ shared_execution_environment_and_factorio_background }}

{{ shared_task_setting }}

## Your Role

Objective:
- produce a complete design that can be parsed and validated,
- meet or exceed the required target production rate per minute after execution,
- and minimize total entity cost among designs that satisfy the task and validation rules.

Scope:
- decide the number and placement of machines, inserters, logistics chests, regular containers, roboports, logistics robots, and power entities;
- when using requester/provider robot logistics, write the executable top-level `logistic_robot_count`;
- return one complete final benchmark-facing design JSON, not a partial subdesign.

## Game Knowledge

{{ knowledge_modules }}

{{ validation_rules }}

Static validation does not enforce a cost limit, and it does not directly verify achieved production rate. Cost is compared among successful designs; production rate is measured by execution and monitoring.

## Output Contract

Return exactly one JSON object matching the public `ProductionLine` schema. Do not return partial drafts or explanatory text outside the JSON object.

Output structure summary:
- Top level: `reasoning`, `task_name`, `target_product`, `target_rate_per_minute`, optional `logistic_robot_count`, `blocks`, `global_entities`, and optional `summary`.
- Each block contains an id, name, optional description, bounding box, machines, inserters, logistics chests, and containers.
- Use concrete supported entity names and real global coordinates.
- `reasoning` and `summary` are explanatory only; every executable decision must also appear in the corresponding JSON field.
```

```
- `logistic_robot_count` must be a positive integer whenever requester/provider logistics are used.
- `request_filters` is required for requester chests and uses positive buffer counts.

## Field Semantics

- `containers` are regular storage chests, not logistics chests.
- `global_entities` contains shared infrastructure such as roboports and power entities.
- `inserter.direction` is interpreted as the drop direction; pickup is behind the inserter.

## Design Guidance

- Prefer designs that are simple, valid, and operationally coherent.
- Complete the recipe chain before optimizing throughput or cost.
- Choose requester/provider roles carefully and power roboports properly.
- Among designs that meet every requirement, prefer the lower-cost one.
```

User Prompt Template

```
{{ shared_task_objective }}

{{ shared_environment }}

{{ shared_recipe_chain }}

{{ shared_previous_turn_feedback_note }}

Return only the final JSON object matching `ProductionLine`.
Do not output more than one top-level JSON object in the same message.
```

D LIMITATIONS

A simulation-to-reality gap remains between Factorio and real factories. Factorio Design Benchmark implements hierarchical design synthesis within one domain, and similarities in task structure and required capabilities do not establish that its scores directly predict performance in other engineering domains.

The task suite covers a limited range of products, target production rates, and map configurations. Broader task coverage and additional independent runs are needed to assess how consistently the observed performance patterns extend to other factory-design settings.

The specification-derived difficulty profile characterizes relative differences within the suite. Its normalized scores depend on the included task distribution and do not define an absolute difficulty scale. The association between these scores and agent success has been assessed on the present suite and requires further validation on expanded task sets.

All reported results use a single-agent iterative protocol with fixed tool access. Alternative agent organizations and tool configurations may affect both absolute performance and model rankings. The experiments also do not compare LLM agents with dedicated optimization solvers or hybrid planning methods.