**28th INTERNATIONAL DEPENDENCY AND STRUCTURE MODELING CONFERENCE, DSM 2026**
MILAN, ITALY, September 29 – October 1, 2026

# Design Structure Matrix Modularization with Large Language Models

Shuo Jiang[1], Jianxi Luo[1]

[1]Department of Systems Engineering, City University of Hong Kong, Hong Kong S.A.R. (China)

Abstract: Design Structure Matrix (DSM) modularization, the task of partitioning system elements into cohesive modules, is a fundamental combinatorial challenge in engineering design. Traditional methods such as Simulated Annealing treat modularization as a pure graph optimization, without access to the engineering context embedded in the system. Large Language Models (LLMs) offer a qualitatively different approach, combining structural reasoning with domain knowledge. In this paper, we apply LLM-based combinatorial optimization (LLM-CO) to DSM modularization across five engineering cases and three backbone LLMs. LLM-CO achieves near-reference quality within 30 iterations without requiring specialized optimization code. We find that while domain knowledge has been shown to benefit DSM sequencing, it consistently impairs performance on more complex DSMs in modularization. We trace this to a semantic misalignment between the LLM's functional priors and the purely structural optimization objective, and formalize this observation as the semantic-alignment hypothesis, a testable condition governing the effectiveness of domain knowledge in LLM-CO. We further examine the influence of edge representation format, objective function explicitness, and solution pool composition on optimization behavior through ablation studies. These findings provide both practical guidance and a framework for deploying LLMs in engineering design optimization.

*Keywords: Artificial Intelligence, Large Language Models, Design Structure Matrix, Combinatorial Optimization, Modularization, Clustering, Systems Engineering*

## 1 Introduction

Engineering systems of growing complexity require structured methods for organizing and managing interdependencies among their constituent elements. The Design Structure Matrix (DSM) has become a widely adopted tool for this purpose, providing a compact matrix representation of pairwise dependencies within a system (Eppinger and Browning, 2012; Steward, 1981). DSM methods have been applied to product architecture, process planning, project management, and organizational design (Langner et al., 2025; Luo, 2015; Roh et al., 2025; Solberg et al., 2025). Many methodological studies have developed around the core analytical tasks: sequencing, partitioning, and modularization (Browning, 2016).

Among these tasks, DSM modularization (also referred to as clustering) is particularly consequential for systems design. Modularization seeks to partition the elements of a DSM into cohesive modules that minimize coupling across module boundaries and maximize interaction density within modules. Good modular architectures enable parallel development, support reuse and upgrade, and reduce the propagation of design changes (Eppinger and Ulrich, 2016; Pimmler and Eppinger, 1994). Thebeau formalized this as an optimization problem via the TotalCost metric, which has been adopted in many studies (Thebeau, 2001). The underlying combinatorial problem is NP-hard: the number of feasible partitions grows as the Bell number B(n), reaching approximately $8\times10^{13}$ for a system with only 17 elements. This exponential growth motivates the development of effective search heuristics.

Existing optimization algorithms, including Simulated Annealing, Genetic Algorithms, and improved clustering heuristics, find near-optimal partitions effectively (Börjesson and Hölttä-Otto, 2012; Thebeau, 2001; Yu et al., 2007). However, these methods treat modularization as a pure graph optimization: they minimize the objectives without reasoning about the engineering meaning of the elements. As a result, they cannot incorporate designer knowledge about functional relationships, system architecture intent, or physical constraints. Recent advances in Large Language Models (LLMs) offer a qualitatively different computational paradigm. Through in-context learning and complex reasoning (Brown et al., 2020), LLMs can process both structural information and natural language descriptions of the elements, in principle combining optimization power with engineering reasoning.

Jiang et al. (2025) demonstrated that LLM-based combinatorial optimization (LLM-CO) can be effectively applied to DSM sequencing, ordering activities to minimize feedback loops. This study found that LLM-CO is competitive with established benchmark methods, and incorporating domain knowledge about activity or component semantics significantly improved solution quality. DSM modularization presents a distinct and more challenging setting for LLM-CO. The two tasks differ fundamentally in their optimization objectives: sequencing minimizes feedback loops, which map directly to workflow dependencies and are semantically accessible to an LLM, while modularization optimizes a structural graph metric with limited inherent semantic content. Whether LLM-CO generalizes effectively to this setting, and whether domain knowledge remains beneficial when semantic alignment between the LLM's priors and the optimization objective cannot be assumed, are open questions that motivate this study.

In this study, we apply LLM-CO to DSM modularization across five engineering cases spanning aerospace, automotive, and mechanical engineering domains, using three backbone LLMs. We address three research questions: (RQ1) Can LLM-CO achieve good modularization quality within a small iteration budget? (RQ2) How does domain knowledge affect

performance, and why does it differ from its role in sequencing? (RQ3) Which input representations and design choices work best in practice with the LLM-CO framework? The main contributions of this paper are: (1) We demonstrate LLM-CO as a viable approach for DSM modularization, achieving near-reference quality within 30 iterations across five engineering cases and three backbone LLMs, without requiring specialized optimization code. (2) We identify a knowledge paradox: domain knowledge, beneficial in DSM sequencing, systematically impairs modularization performance on more complex DSMs. We attribute this to semantic misalignment between the LLM's functional priors and the structural optimization objective, and propose this as a testable condition for knowledge effectiveness in LLM-CO more broadly. (3) Through ablation studies on input representation, objective formulation, and solution pool design, we provide practical guidance for deploying LLM-CO in engineering design optimization.

The remainder of this paper is organized as follows. Section 2 reviews background on DSM modularization and LLMs for combinatorial optimization. Section 3 presents the LLM-CO framework. Section 4 describes the experimental setup. Section 5 presents experimental results and discusses the findings. Section 6 concludes the paper.

## 2 Background

### 2.1 DSM Modularization

The Design Structure Matrix was introduced by Steward (1981) as a square matrix representation of directed dependencies among system elements. A non-zero entry $DSM_{ij}$ indicates that element i receives input from element j, with the entry value representing the strength of that dependency. DSMs come in three principal types depending on what the rows and columns represent: activity-based DSMs encode information flows between design activities; parameter-based DSMs represent functional dependencies among design parameters; and component-based DSMs capture physical interfaces between hardware components (Browning, 2001; Eppinger and Browning, 2012). The present study includes all three types.

Pimmler and Eppinger (1994) established a framework for quantifying interactions across spatial, energy, information, and material flows, providing the theoretical basis for assigning numerical interaction strengths in a DSM. Thebeau (2001) built on this to formalize modularization as an optimization problem, introducing TotalCost as the objective function and proposing a stochastic hill-climbing algorithm to optimize this objective. Later work has explored broader algorithmic approaches. For example, Yu et al. (2007) applied genetic algorithms to exploit global search capabilities. Börjesson and Hölttä-Otto (2012) developed an improved clustering algorithm that avoids some local optima of the original hill-climbing method. These algorithms have been validated on a range of industrial DSMs and represent the current state of practice. However, such methods treat modularization as a pure graph optimization without incorporating engineering knowledge.

### 2.2 LLMs for Combinatorial Optimization

Combinatorial optimization has a long history in operations research and computer science, ranging from exact methods (e.g., branch-and-bound, dynamic programming) to metaheuristics (e.g., simulated annealing, evolutionary algorithms). Recent work has explored a new paradigm, using LLMs as optimizers. Yang et al. (2024) introduced OPRO, demonstrating that LLMs can optimize objective functions by iteratively refining candidate solutions described in natural language. Given a prompt containing previously evaluated solutions ranked by quality, the LLM generates improved candidates by extrapolating patterns from the examples. This approach requires no gradient computation or explicit solver code, relying instead on the LLM's pattern-completion capabilities. FunSearch (Romera-Paredes et al., 2024) extended this idea to mathematical discovery, using LLMs to search over program space for functions that improve on known solutions to hard combinatorial problems. Their results demonstrated that LLM-guided search can discover state-of-the-art solutions on problems including cap set construction and bin packing. These approaches share a common structure: an LLM generates candidate solutions, which are evaluated by an external scorer, with the best solutions fed back as in-context examples for the next generation round.

In the DSM domain, two recent lines of work apply LLMs to DSM tasks. Koh (2024) introduced Auto-DSM, demonstrating that LLMs can generate DSM structures from natural language project descriptions, a generation task rather than optimization. A follow-up study (Koh, 2026) further evaluated LLM-based extraction of indirect design dependencies, examining how writing style and entity naming affect retrieval accuracy across five LLMs. Together, these studies establish that LLMs can interpret and produce DSM content from text, though they do not address modularization. In addition, Jiang et al., (2025) applied LLM to DSM sequencing, finding that LLM-CO is competitive with established methods and that domain knowledge about activity semantics consistently improved solution quality. The present work extends these findings to DSM modularization, where the objective function differs fundamentally in its semantic interpretability.

# 3 LLM-CO Framework for DSM Modularization

## 3.1 Problem Formulation

Given a DSM with n elements represented as a weighted directed graph, the modularization problem seeks a partition $\pi$ of elements into $M$ modules ($2 \leq M \leq N$) that minimizes a structural cost objective. Because the global optimum is computationally intractable for large n, we establish a reference benchmark by running Simulated Annealing (SA) with 10,000 independent random restarts and taking the best solution found; we refer to this as the SA reference (computational details in Appendix A.3). The SA reference is not provably optimal but represents the best-known solution under this metric and provides a stable, reproducible reference for comparison.

## 3.2 LLM-CO Framework

Figure 1 illustrates the proposed LLM-CO framework. The iterative procedure consists of four stages: **(1) Initialize and build prompt** (input: the DSM edge list, the current solution pool, and the knowledge flag k; output: a structured text prompt). At the first iteration, each element is assigned to its own singleton module, yielding n modules of size 1. The current solution pool is then encoded into a structured prompt containing the DSM representation, the ranked prior solutions, and optionally domain knowledge about the elements. **(2) Query LLM and validate** (input: the structured prompt; output: a validated candidate partition, or a rejection that skips the iteration). The LLM is queried to generate a new candidate partition, expressed as a mapping from element identifiers to module labels; the response is checked for structural validity, requiring all elements to be assigned to exactly one module with at least two modules total. **(3) Evaluation** (input: the validated partition and the weighted DSM; output: its cost value). The cost objective is computed for the validated partition. **(4) Update solution pool** (input: the evaluated candidate and the current pool; output: the updated pool and the best-ever solution). The pool retains the top-ranked solutions found so far, supplemented by randomly sampled solutions from the history to maintain diversity; the best-ever solution is tracked separately. This loop repeats for a fixed number of iterations, with the best solution found across all iterations returned as the final output.

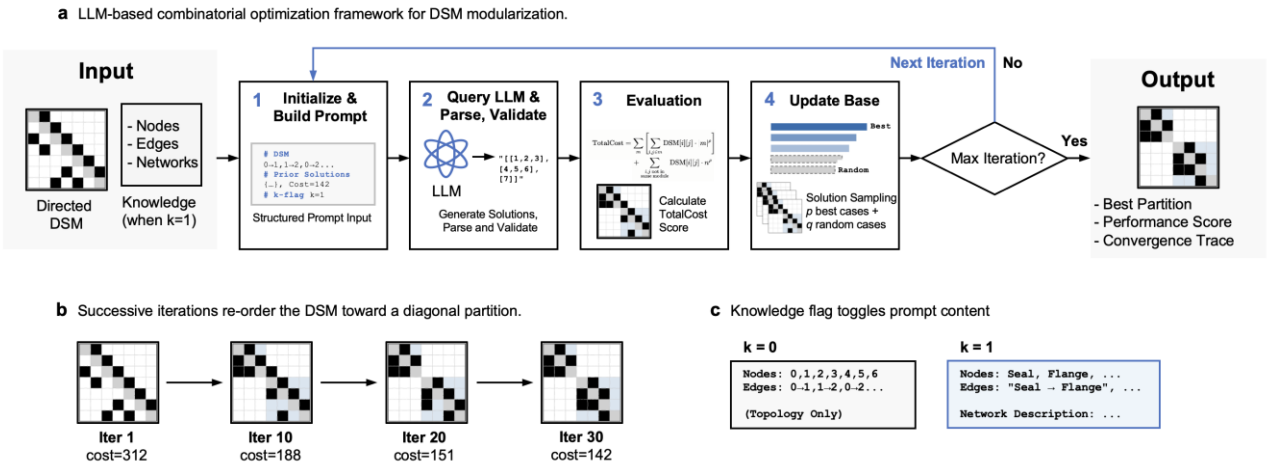


Figure 1. LLM-based combinatorial optimization framework for DSM modularization. (a) The iterative loop consists of four stages: prompt construction, LLM query and validation, evaluation, and solution pool update. (b) Successive iterations re-order the DSM toward a block-diagonal partition. (c) Knowledge flag k controls prompt content.

Each prompt is structured around three components. The first is the DSM representation. The DSM is encoded as a directed edge list: a set of (i, j, weight) triples indicating that element i receives input from element j with a given strength. Node identifiers are shuffled randomly at each iteration to prevent position bias. This format was selected based on ablation results, as it provides complete directional information while remaining compact and reliably parseable by all three LLMs tested. The second component is the solution base. Previously found partitions are presented in ranked order from best to worst, providing the LLM with concrete examples from which to infer the optimization direction. In our default configuration, the pool contains p=5 best solutions found so far and q=5 randomly sampled solutions, balancing exploitation of high-quality regions and exploration of the broader solution space. Each solution is presented as a module-assignment list together with its objective value. The third component is a knowledge flag that determines whether engineering semantic content is included in the prompt. Under the knowledge-off condition (k=0), node identifiers are replaced with anonymized labels (e.g., N1, N2, ...) and no natural language descriptions are provided, so the LLM receives only topological information. Under the knowledge-on condition (k=1), each node is identified by its engineering name and accompanied by a brief functional description. The comparison between k=0 and k=1 forms the central experimental contrast of this study. Appendix A.1 provides more details about the prompt design.

## 4 Experimental Setup

### 4.1 DSM Cases

We evaluate LLM-CO on five engineering DSM cases, summarized in Table 1, where N denotes the number of nodes, E the number of directed edges, and Density = E/(N×(N−1)). The cases range from 12 to 19 elements, spanning activity-based, parameter-based, and component-based DSM types across aerospace, automotive, and mechanical engineering domains. All five cases were collected from peer-reviewed literature. For each case, three types of information were extracted from the original reference: the names of all system elements, the edge list derived from the adjacency matrix, and a one-sentence natural language description of each element's generic main function within the system. This functional description characterizes the element itself, independent of its role in any specific pairwise dependency. All pairwise dependency information is instead conveyed exclusively by the edge list. These cases extend the dataset used in the study on LLM-CO for DSM sequencing (Jiang et al., 2025).

Table 1. Five Benchmark DSM Cases

| Case | N | Edges | Density | DSM Type | Domain | Source |
|---|---|---|---|---|---|---|
| Unmanned Combat Aerial Vehicle (UCAV) | 12 | 52 | 0.40 | Activity | Aerospace | (Browning, 1998) |
| Kodak Cartridge | 13 | 49 | 0.32 | Parameter | Automotive | (Eppinger and Ulrich, 2016) |
| Brake System | 14 | 35 | 0.19 | Activity | Automotive | (Black et al., 1990) |
| Heat Exchanger (HeatEx) | 17 | 43 | 0.16 | Parameter | Mechanical Engineering | (Amen et al., 1999) |
| Helicopter | 19 | 109 | 0.32 | Component | Aerospace | (Clarkson et al., 2004) |

### 4.2 Experimental Settings

We evaluate three backbone LLMs: Claude-Sonnet-4.6 (hereafter **Claude**; Anthropic (2026)), GPT-5.2 (hereafter **GPT**; OpenAI, (2025)), Qwen-3.5-Plus (hereafter **Qwen**; Qwen Team, (2026)). These models represent distinct architectures and training regimes, allowing us to assess the robustness of findings across model families. For each combination of model, DSM case, and knowledge condition (k=0 and k=1), we run 10 independent trials of 30 iterations. All experiments use the default solution pool configuration (p=5 best, q=5 random) and directed edge list input format unless otherwise specified. Temperature is set to 1.0 for all models to encourage solution diversity. More implementation details are provided in Appendix A.

### 4.3 Evaluation Metrics

We use two metrics to evaluate modularization quality. TotalCost (Thebeau, 2001) is the primary optimization objective:

$$\text{TotalCost} = \sum_{m} \left[ \sum_{i,j \in m} \text{DSM}_{\text{ij}} \cdot |m|^{\rho} \right] + \sum_{\substack{i,j \text{ not in} \\ \text{same module}}} \text{DSM}_{\text{ij}} \cdot n^{\rho}$$

where $\text{DSM}_{\text{ij}}$ is the interaction weight between elements i and j, $|m|$ is the size of module $m$, $n$ is the total number of elements, and $\rho$ is a size-penalty exponent (set to 1 throughout). Lower TotalCost is better: the formulation rewards dense intra-module interactions (multiplied by $|m| < n$) and penalizes cross-module interactions.

Clustering Efficiency (CE) is a secondary metric defined as the fraction of total weighted interactions that fall within modules: $\text{CE} = \sum_{intra} \text{DSM}_{\text{ij}} / \sum_{\text{all}} \text{DSM}_{\text{ij}}$. CE ranges from 0 to 1, and higher values indicate better modularization. CE is reported alongside TotalCost in the experimental results as a complementary measure of solution quality.

Unless otherwise noted, all metrics are aggregated as mean ± standard deviation across 10 independent runs. For convergence analysis, we additionally report Gap% = (TotalCost − SA reference) / SA reference × 100%, where a Gap% of 0% indicates that the run matched the SA reference.

### 4.4 Ablation Conditions

To understand what shapes LLM-CO performance across key design choices, we conduct ablation studies on three dimensions, all using the UCAV case with the Claude model. Domain knowledge is included in all ablation conditions. This case and model were selected because they represent a mid-complexity setting where all three ablation dimensions produce observable performance differences.

For solution pool design, we compare three variants: the balanced baseline (p=5 best, q=5 random), an exploitation-only variant (p=5 best, q=0), and an exploration-only variant (p=0, q=5 random). For objective formulation, we compare the baseline prompt against a simple prompt with the explicit TotalCost formula removed, retaining only the natural language task description and ranked solution examples. For input representation, we compare the directed edge list baseline against an adjacency matrix, an undirected edge list (directionality removed), and a natural language description of dependencies.

## 5 Results and Discussion

### 5.1 Modularization Quality and Convergence Behavior

Table 2 summarizes mean TotalCost and Clustering Efficiency (CE) across 10 independent runs for all three models and knowledge conditions. Claude under k=0 achieves the most consistent performance: on the largest case (Helicopter, N=19), all 10 runs match the SA reference in TotalCost (1371±0), while CE (0.675±0.011) exceeds the SA reference value of 0.514, indicating that LLM-CO identifies equally optimal partitions with higher intra-module interaction density. Notably, incorporating domain knowledge does not consistently improve performance here, in contrast to the findings in DSM sequencing where knowledge was uniformly beneficial (Jiang et al., 2025). Across the three intermediate cases (Kodak, Brake, Heat Exchanger), Claude k=0 remains within 1.3% of the SA reference. GPT and Qwen achieve competitive results on smaller cases but exhibit substantially higher run-to-run variance, particularly on UCAV and Helicopter. Across all three models, the results confirm that LLM-CO is a viable approach for DSM modularization without writing any solver code.

Table 2. Performance Across Models, Knowledge Conditions, and DSM Cases

| **Models** | **Knowledge** | **Performance (Total Cost & Clustering Efficiency)** | | | | |
|---|---|---|---|---|---|---|
| | | UCAV | Kodak | Brake | HeatEx | Helicopter |
| Claude-Sonnet-4.6 @30 | with (k=1) | **450.0±0.0** (0.56±0.00) | 437.8±0.4 (0.64±0.01) | 292.4±3.7 (0.63±0.00) | 433.8±7.4 (0.70±0.07) | 1420.8±36.6 (0.59±0.10) |
| | without (k=0) | 454.8±10.5 (0.53±0.05) | **436.7±2.2** (0.59±0.07) | **291.6±3.2** (0.63±0.00) | **412.0±9.1** (0.73±0.04) | **1371.0±0.0** (0.68±0.01) |
| GPT-5.2 @30 | with (k=1) | 517.6±53.3 (0.72±0.17) | 462.2±25.1 (0.74±0.11) | 300.4±13.4 (0.66±0.07) | 448.0±0.0 (0.65±0.00) | 1391.9±30.0 (0.64±0.06) |
| | without (k=0) | 503.1±42.4 (0.68±0.10) | 461.3±10.1 (0.79±0.10) | 290.0±0.0 (0.63±0.00) | 457.9±24.5 (0.77±0.09) | 1481.1±176.7 (0.70±0.08) |
| Qwen-3.5-Plus @30 | with (k=1) | 538.9±56.7 (0.81±0.17) | 444.8±8.3 (0.66±0.03) | 298.3±0.5 (0.62±0.01) | 441.8±11.2 (0.64±0.03) | 1543.8±44.0 (0.41±0.05) |
| | without (k=0) | 526.0±38.7 (0.72±0.15) | 442.4±6.7 (0.53±0.05) | 298.8±15.8 (0.66±0.07) | 483.0±58.4 (0.83±0.07) | 1406.9±42.6 (0.65±0.06) |
| SA reference (10K Best) | | 446 (0.44) | 434 (0.51) | 290 (0.63) | 407 (0.74) | 1371 (0.51) |

Figure 2 shows the convergence behavior of all three models across all five cases. Claude converges rapidly and stably: Gap% drops sharply within the first five iterations and approaches a plateau well before iteration 30, with narrow standard deviation bands across all cases. GPT and Qwen present a contrasting picture: convergence is slower, bands are substantially wider, and several panels show curves still declining at iteration 30, particularly on larger cases. This suggests that while a 30-iteration budget is sufficient for Claude, GPT and Qwen may benefit from additional iterations on more complex DSMs. The difference in convergence stability across models is consistent with the variance patterns reported in Table 2, and reinforces Claude as the most reliable backbone for DSM modularization within the tested iteration budget.

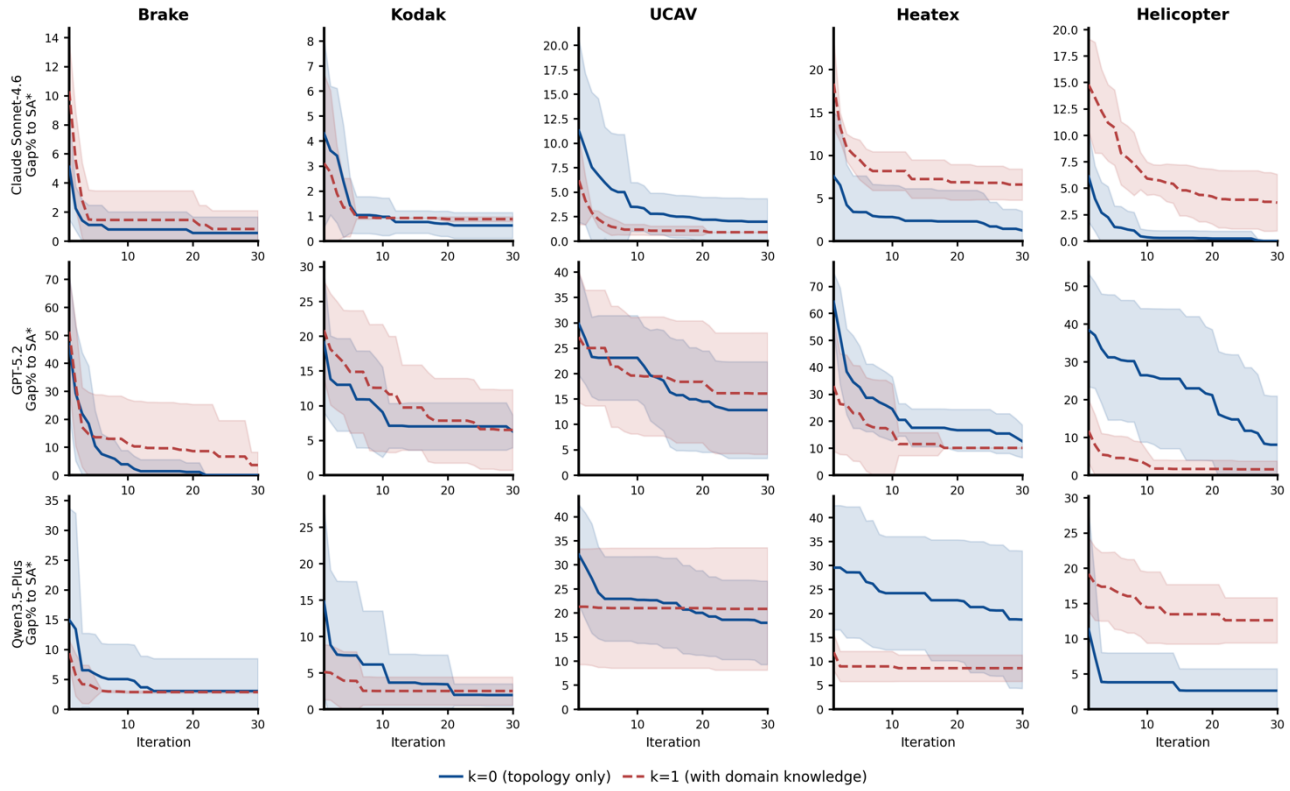


Figure 2. Convergence Behavior of LLM-CO Across Three LLMs and Five DSM Cases

### 5.2 The Knowledge Paradox: When Domain Information Hurts

The effect of domain knowledge on performance depends strongly on both DSM complexity and model capability, as shown jointly by the final Gap% results (Figure 3) and the convergence trajectories (Figure 2). For Claude, the pattern is clear: domain knowledge shows mixed or negligible effects on the three simpler cases, with a marginal improvement on UCAV (Gap% of 0.9% with vs. 2.0% without domain knowledge) and essentially no change on Kodak and Brake. On the two larger cases, however, domain knowledge consistently and substantially impairs performance, with HeatEx incurring a 5.4% penalty and Helicopter a 3.6% penalty. On the Helicopter case, Claude k=0 achieves 10 out of 10 SA-reference-matching runs while k=1 yields a mean Gap% of 3.6% with zero runs reaching the reference. The convergence curves in Figure 2 reinforce this finding: for Claude, the separation between k=0 and k=1 is established within the first few iterations and maintained throughout, indicating that knowledge shapes the search trajectory from the outset rather than imposing a transient initial penalty.

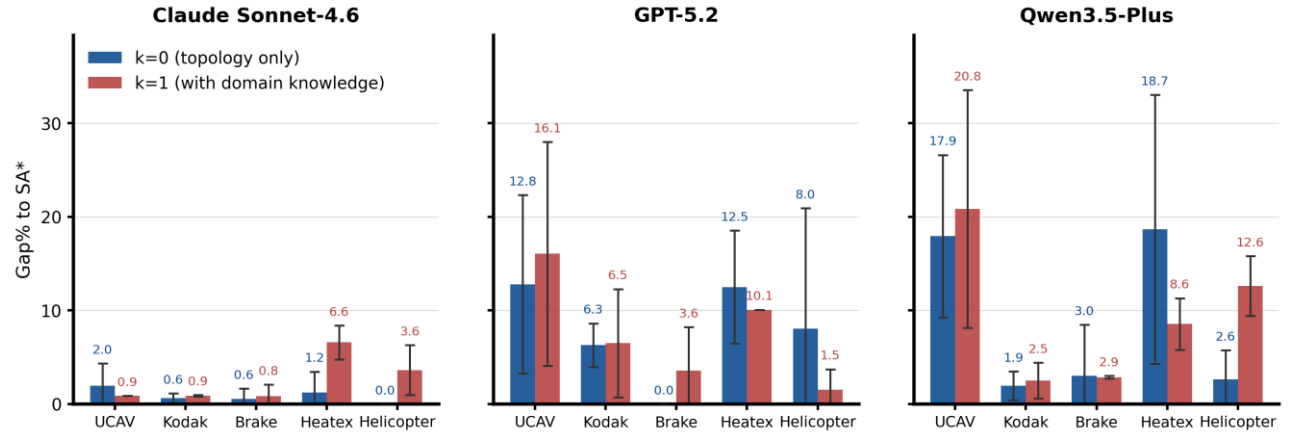


Figure 3. Modularization Quality with and without Domain Knowledge

For GPT and Qwen, the convergence trajectories in Figure 2 reveal a more nuanced pattern: k=1 tends to reach lower Gap% in early iterations, suggesting that domain knowledge provides a useful shortcut when the model cannot yet identify

high-quality structural partitions on its own. However, k=0 catches up or surpasses k=1 in later iterations across most cases, indicating that semantic priors ultimately constrain rather than guide the search. This interaction between knowledge and model capability supports a capability-moderation interpretation: for stronger models like Claude, structural optimization under k=0 is reliable enough that semantic priors serve only as a distraction; for weaker models, knowledge offers an early advantage that diminishes as the solution pool accumulates informative examples.

We attribute the impairment to semantic misalignment between the LLM's domain priors and the optimization objective. When node names are visible, the LLM applies engineering intuitions about functional grouping that are intuitively reasonable but diverge from the structural optimum. For simpler DSMs, functional and structural groupings tend to coincide and the effect is negligible; for more complex DSMs, the two diverge and the penalty becomes consistent. This stands in contrast to DSM sequencing (Jiang et al., 2025), where minimizing feedback loops aligns directly with engineering process logic and domain knowledge consistently helps.

### 5.3 Ablation Analysis

Figure 4 examines three design dimensions of the LLM-CO framework, all using UCAV with Claude under k=1. The solution pool design has a clear effect on both final quality and convergence stability (Figure 4a). The balanced pool (5 best + 5 random) achieves the lowest final Gap% with the narrowest standard deviation band, outperforming both the exploitation-only (best-only) and exploration-only (random-only) variants. The exploitation-only variant converges quickly in early iterations but with higher run-to-run variance, while the exploration-only variant shows slower early improvement in the absence of high-quality exemplars. These results suggest that maintaining both exploitation of high-quality solutions and exploration of the broader solution space is important for stable and effective LLM-CO.

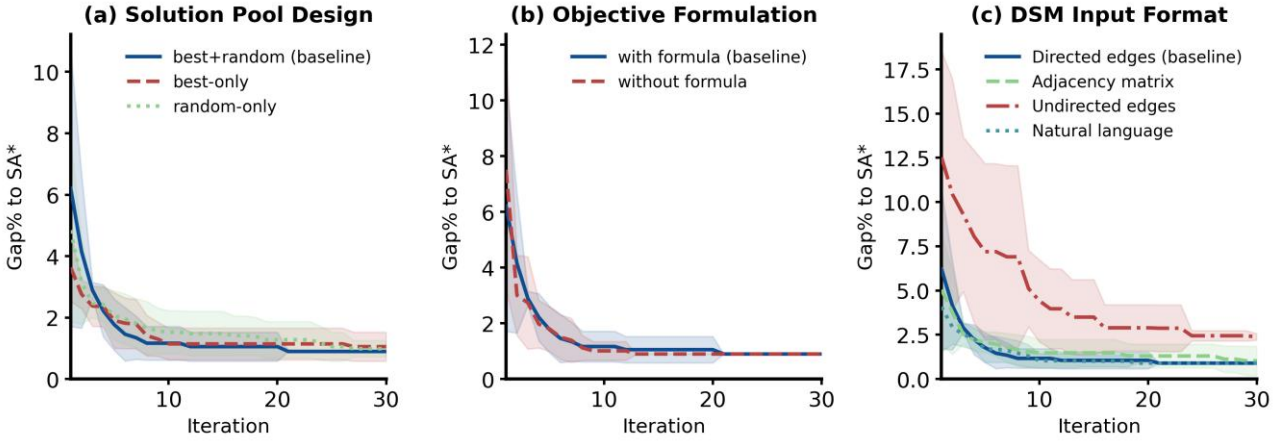


Figure 4. Ablation convergence behavior. (a) Solution pool design. (b) Objective formulation. (c) DSM input format.

Removing the explicit objective formula from the prompt has no measurable effect: the two conditions produce near-identical convergence curves throughout all 30 iterations (Figure 4b). This indicates that the LLM does not use the formula for numerical evaluation but instead infers the optimization direction from the ranked solution examples alone, consistent with in-context learning as implicit optimization (Yang et al., 2024). A practical consequence is that LLM-CO can be applied to problems where the objective is difficult to specify analytically, provided that candidate solutions can be evaluated and ranked externally.

Among the input representation variants (Figure 4c), the directed edge list, adjacency matrix, and natural language description all converge to similarly low Gap% by iteration 30, but the undirected edge list remains clearly worse throughout. The comparable performance of the natural language format is noteworthy: it indicates that verbalized relational statements preserve the directional information the LLM requires, so that dependency information embedded in textual design documentation could in principle be provided to LLM-CO directly, without first being converted into a formal graph encoding. Its main drawback is verbosity, as token count grows faster than that of the compact edge list with the number of edges. Removing directionality from the edge representation is the single most damaging design choice tested, demonstrating that dependency direction is a critical signal for LLM-CO: knowing which element depends on which, rather than merely that two elements are connected, substantially improves solution quality. The directed edge list is recommended as the default format for its combination of directional completeness, compactness, and parsing reliability.

## 6 Concluding Remarks

This paper presents a systematic investigation of LLM-based combinatorial optimization for DSM modularization across five engineering cases and three backbone LLMs. LLM-CO achieves near-reference quality within 30 iterations without solver code or gradient computation, with Claude-Sonnet-4.6 demonstrating the most consistent performance across all cases. The central finding is a knowledge paradox: domain knowledge, beneficial in LLM-based DSM sequencing, systematically impairs modularization performance on more complex DSMs. We attribute this to semantic misalignment between the LLM's functional priors and the purely structural optimization objective. Across models, we find that this paradox is most clearly observable in stronger models capable of reliable structural optimization, while less capable models show an early advantage from domain knowledge that diminishes as the solution pool matures. The absolute performance figures reported here are inevitably bound to the model versions tested, whereas the core contributions of this study are properties of the approach itself, not of any particular model. The framework is model-agnostic, and the full protocol is documented in Appendix, allowing the study to be replicated on newer models, including recent agentic and multi-agent systems.

This study has several limitations. The benchmark cases cover DSMs with at most 19 elements, and scalability to larger instances remains an open question. The semantic misalignment effect is expected to be more prevalent on larger, industrial-scale DSMs, where functional and structural groupings diverge more strongly. A natural next step is therefore to test LLM-CO on such systems, for example through hierarchical decomposition of the DSM or retrieval-based presentation of the solution pool to keep prompt length manageable. The SA reference is not provably optimal for the larger cases, and additional model families beyond the three tested may exhibit different knowledge sensitivity patterns. Different DSM types may exhibit different patterns of semantic alignment, and the knowledge effect may vary accordingly. For instance, Sosa et al. (2004) demonstrated empirically that misalignment between component architecture and organizational structure can be substantial and difficult to detect, suggesting that domain knowledge about physical interfaces may be more tacit and harder for LLMs to leverage effectively than knowledge about activities or parameters. Systematically investigating this across DSM types requires broader empirical coverage than the present study provides, and a controlled follow-up study varying DSM type while holding size and model constant would help isolate this effect. Finally, this study evaluates modularization quality using TotalCost and Clustering Efficiency, which are commonly used in the engineering design community; future work should examine whether findings generalize to alternative metrics from the network science community, such as the modularity metric described by Leicht and Newman (2008).

Together, the findings presented in this paper establish LLM-CO as a viable and flexible approach for engineering design optimization, one that requires no gradient computation or solver code and can be adapted to new objectives by updating the evaluation function alone. More broadly, the semantic-alignment hypothesis offers a testable principle for anticipating when domain knowledge will help or hurt in LLM-based optimization, with implications extending beyond DSMs to a wider class of combinatorial problems in engineering design.

## References

Amen, R., Rask, I., Sunnersjö, S., 1999. Matching Design Tasks to Knowledge-Based Software Tools: When Intuition Does Not Suffice, in: Proceedings of the ASME International Design Engineering Technical Conferences and Computers and Information in Engineering Conference (IDETC/CIE 1999). ASME, pp. 1165–1174.

Anthropic, 2026. Claude Sonnet 4.6 System Card [WWW Document]. URL https://www.anthropic.com/news/claude-sonnet-4-6

Black, T.A., Fine, C.H., Sachs, E.M., 1990. A Method for Systems Design Using Precedence Relationships: An Application to Automotive Brake Systems (Working Paper). Sloan School of Management, Massachusetts Institute of Technology.

Börjesson, F., Hölttä-Otto, K., 2012. Improved Clustering Algorithm for Design Structure Matrix, in: Proceedings of the ASME 2012 International Design Engineering Technical Conferences & Computers and Information in Engineering Conference (IDETC/CIE 2012). ASME, pp. 921–930. https://doi.org/10.1115/DETC2012-70076

Brown, T.B., Mann, B., Ryder, N., Subbiah, M., Kaplan, J.D., Dhariwal, P., Neelakantan, A., Shyam, P., Sastry, G., Askell, A., Agarwal, S., Herbert-Voss, A., Krueger, G., Henighan, T., Child, R., Ramesh, A., Ziegler, D.M., Wu, J., Winter, C., Hesse, C., Chen, M., Sigler, E., Litwin, M., Gray, S., Chess, B., Clark, J., Berner, C., McCandlish, S., Radford, A., Sutskever, I., Amodei, D., 2020. Language Models are Few-Shot Learners, in: Advances in Neural Information Processing Systems. Curran Associates, Inc., pp. 1877–1901.

Browning, T.R., 2016. Design Structure Matrix Extensions and Innovations: A Survey and New Opportunities. IEEE Trans. Eng. Manag. 63, 27–52. https://doi.org/10.1109/TEM.2015.2491283

Browning, T.R., 2001. Applying the Design Structure Matrix to System Decomposition and Integration Problems: A Review and New Directions. IEEE Trans. Eng. Manag. 48, 292–306. https://doi.org/10.1109/17.946528

Browning, T.R., 1998. Modeling and Analyzing Cost, Schedule, and Performance in Complex System Product Development (PhD Thesis). Sloan School of Management, Massachusetts Institute of Technology, Cambridge, MA.

Clarkson, P.J., Simons, C., Eckert, C., 2004. Predicting Change Propagation in Complex Design. J. Mech. Des. 126, 788–797. https://doi.org/10.1115/1.1765117

Eppinger, S.D., Browning, T.R., 2012. Design Structure Matrix Methods and Applications. MIT Press, Cambridge, MA.

Eppinger, S.D., Ulrich, K.T., 2016. Product Design and Development, 6th ed. McGraw-Hill Education, New York, NY.

Jiang, S., Xie, M., Luo, J., 2025. Large Language Models for Combinatorial Optimization of Design Structure Matrix. Proc. Des. Soc. 5. https://doi.org/10.1017/pds.2025.10234

Koh, E.C.Y., 2026. From Text to DSM: Evaluating the Impact of Writing Style and Entity Naming on LLM-Based Retrieval of Asymmetrical Indirect Design Dependencies. Res. Eng. Des. 37, 13. https://doi.org/10.1007/s00163-026-00476-2

Koh, E.C.Y., 2024. Auto-DSM: Using a Large Language Model to Generate a Design Structure Matrix. Nat. Lang. Process. J. 9, 100103. https://doi.org/10.1016/j.nlp.2024.100103

Langner, C., Paliyenko, Y., Roth, D., Kreimeyer, M., 2025. Utilizing DSM and SysML for Modeling Data Flows in Complex Networks – A Case Study on Autonomous Public Transportation, in: Proceedings of the 27th International DSM Conference (DSM 2025), DS 141. Hoboken, NJ, USA, pp. 11–20. https://doi.org/10.35199/dsm2025.02

Leicht, E.A., Newman, M.E.J., 2008. Community Structure in Directed Networks. Phys. Rev. Lett. 100, 118703. https://doi.org/10.1103/PhysRevLett.100.118703

Luo, J., 2015. A simulation-based method to evaluate the impact of product architecture on product evolvability. Res. Eng. Des. 26, 355–371. https://doi.org/10.1007/s00163-015-0202-3

OpenAI, 2025. GPT-5.2 System Card [WWW Document]. URL https://openai.com/index/gpt-5-system-card-update-gpt-5-2/

Pimmler, T.U., Eppinger, S.D., 1994. Integration Analysis of Product Decompositions, in: Proceedings of the ASME Design Theory and Methodology Conference. ASME, pp. 343–351. https://doi.org/10.1115/DETC1994-0034

Qwen Team, 2026. Qwen3.5: Towards Native Multimodal Agents [WWW Document]. URL https://qwen.ai/blog?id=qwen3.5

Roh, H., Etzenbach, L., Oltramare, A., Norheim, J., de Weck, O., 2025. Factored Dependency Structure Matrix for Representation of Multi-Connection Systems, in: Proceedings of the 27th International DSM Conference (DSM 2025), DS 141. Hoboken, NJ, USA, pp. 31–40. https://doi.org/10.35199/dsm2025.04

Romera-Paredes, B., Barekatain, M., Novikov, A., Balog, M., Kumar, M.P., Dupont, E., Ruiz, F.J.R., Ellenberg, J.S., Wang, P., Fawzi, O., Kohli, P., Fawzi, A., 2024. Mathematical Discoveries from Program Search with Large Language Models. Nature 625, 468–475. https://doi.org/10.1038/s41586-023-06924-6

Solberg, R., Yassine, A., Worren, N., Soldal, K., Christiansen, T., 2025. DSMs for Organization Design: Incorporating Additional Criteria in Clustering Algorithms, in: Proceedings of the 27th International DSM Conference (DSM 2025), DS 141. Hoboken, NJ, USA, pp. 89–98. https://doi.org/10.35199/dsm2025.10

Sosa, M.E., Eppinger, S.D., Rowles, C.M., 2004. The Misalignment of Product Architecture and Organizational Structure in Complex Product Development. Manag. Sci. 50, 1674–1689. https://doi.org/10.1287/mnsc.1040.0289

Steward, D.V., 1981. The Design Structure System: A Method for Managing the Design of Complex Systems. IEEE Trans. Eng. Manag. EM-28, 71–74. https://doi.org/10.1109/TEM.1981.6448589

Thebeau, R.E., 2001. Knowledge Management of System Interfaces and Interactions for Product Development Processes (Master's Thesis). Massachusetts Institute of Technology, Cambridge, MA.

Yang, C., Wang, X., Lu, Y., Liu, H., Le, Q.V., Zhou, D., Chen, X., 2024. Large Language Models as Optimizers, in: International Conference on Learning Representations (ICLR 2024).

Yu, T.-L., Yassine, A.A., Goldberg, D.E., 2007. An Information Theoretic Method for Developing Modular Architectures Using Genetic Algorithms. Res. Eng. Des. 18, 91–109. https://doi.org/10.1007/s00163-007-0030-1

**Contact: S. Jiang**, Department of Systems Engineering, City University of Hong Kong, 83 Tat Chee Avenue, Kowloon, Hong Kong SAR, shuo.jiang@cityu.edu.hk; J. Luo, Department of Systems Engineering, City University of Hong Kong, 83 Tat Chee Avenue, Kowloon, Hong Kong SAR, jianxi.luo@cityu.edu.hk.

# Appendix A: Implementation Details

### A.1 Prompt Details

Each LLM query consists of a system message and a user message. The system message establishes the task context: the LLM is instructed to act as a DSM modularization expert whose goal is to find a partition of system elements that minimizes the structural cost objective. The user message contains three structured blocks.

Block 1: DSM description. Under k=0, nodes are labeled with anonymized identifiers (e.g., N01, N02, ...) in randomized order. Under k=1, each node is presented with its engineering name, the element label used in the original source publication, and a one-sentence functional description of the element's generic main role in the system (e.g., 'N01 (Fuel System): Stores and delivers fuel to the propulsion subsystem'), rather than a description of its involvement in any individual dependency. The DSM structure is given as a directed edge list: 'N03 -> N01 (weight: 4)', one edge per line. Edges are shuffled randomly at each iteration to prevent the LLM from relying on list position as a proxy for node importance.

Block 2: Solution base. Up to 10 previously evaluated partitions are presented, sorted in ascending order of TotalCost (best first). Each solution is shown as a mapping from node to module label (e.g., 'N01: M2, N02: M1, N03: M2, ...') followed by its TotalCost value. In the baseline configuration, the pool contains the 5 best solutions found so far and 5 randomly sampled solutions from the full history. In the objective formulation ablation (Section 4.4), a formula variant additionally states the TotalCost formula explicitly; in the no-formula variant, only the ranked examples are provided.

Block 3: Generation instruction. The LLM is instructed to generate a new partition that achieves lower TotalCost than the best solution shown, to output the result in a parseable format (JSON dictionary mapping node label to module label), and not to copy any of the provided solutions verbatim.

### A.2 Parsing and Validation

The LLM response is parsed by extracting the first JSON-like block from the output using a regular expression. The extracted mapping is validated against three criteria: (1) all n node identifiers are present and assigned to exactly one module; (2) at least 2 distinct module labels are used; and (3) all values are string labels rather than numerical indices. If any criterion fails, the response is discarded and the iteration is recorded as invalid. Module labels are then canonicalized to consecutive integers (M1, M2, ..., MK) before TotalCost evaluation.

### A.3 SA Reference Computation

The SA reference for each case is computed using simulated annealing with a geometric cooling schedule ($\alpha$=0.9, 150 temperature steps). Each run starts from a random initial partition; at each step, a randomly selected element is proposed for reassignment to an existing or new module, with the move accepted probabilistically according to the SA acceptance criterion. A total of 10,000 independent random restarts are executed, and the SA reference is the minimum TotalCost observed across all restarts.

### A.4 Hyperparameter Settings

Table A1 summarizes the key hyperparameters used in all experiments. The values listed correspond to the default configuration used in the experiments and are held constant across all models, cases, and knowledge conditions unless explicitly varied as part of the ablation study.

Table A1. LLM-CO hyperparameter settings.

| Hyperparameter | Value |
|---|---|
| Number of iterations | 30 |
| Number of independent runs | 10 per condition |
| Solution pool: best kept | 5 |
| Solution pool: random added | 5 |
| TotalCost exponent ($\rho$) | 1 |
| LLM temperature | 1.0 |
| Default input format | Directed edge list |
| Default knowledge condition | k=0 and k=1 (both reported) |