How Little is Enough? Discovering Minimal Prompts through Analogical Abstraction for LLM Reasoning

Anonymous ACL submission

Abstract

Large language models (LLMs) often solve complex tasks with prompts that far exceed the information strictly required for correct reasoning. This raises a fundamental question: what is the minimal structural description sufficient to elicit correct reasoning in an LLM? We propose a diagnostic framework to empirically probe this question using analogical abstraction. Rather than treating prompt compression as an efficiency problem, we view it as a tool for measuring the structural sufficiency threshold of model reasoning. Using code generation tasks with executable unit tests, we iteratively replace original problem descriptions with progressively abstracted analogical descriptions, testing whether functional equivalence is preserved. This allows us to identify the point at which removing structural information causes reasoning failure. Our results show that (i) LLMs can often reconstruct correct solutions from highly abstract analogical descriptions, (ii) different abstraction styles exhibit distinct failure modes, and (iii) models vary substantially in their sensitivity to structural information. These findings suggest that analogical abstraction provides a principled probe for analyzing the minimal conditions under which LLMs retain structured reasoning, offering insights into how pretrained models internalize and retrieve task structure.

1 Introduction

Analogy is a central mechanism in human reasoning that enables complex problems to be reformulated in simpler terms by preserving relational structure while discarding surface detail (Gentner, 1983; Bassok and Holyoak, 1989; Holyoak and Thagard, 1996). Through analogical mapping, humans reinterpret novel problems by aligning them with structurally similar past experiences, allowing reasoning to operate over abstract relational patterns rather than concrete descriptions (Hummel and Holyoak, 1997; Gentner and Forbus, 2011). Such abstractions support effective problem solving across diverse domains, including mathematical reasoning, planning, and decision making (Gentner and Markman, 1997; Gick and Holyoak, 1980; Jamrozik et al., 2016).

Despite its importance in human cognition, analogical abstraction remains underexplored in large language models (LLMs). Recent studies demonstrate that LLMs can leverage analogies when these are explicitly provided, for instance through few-shot prompts containing analogical exemplars. However, in these settings analogy functions primarily as an *input condition* or prompting strategy, rather than as a *generative capacity* of the model itself. As a result, it remains unclear whether an LLM can independently produce an analogical abstraction that is structurally sufficient to elicit the same reasoning behavior as the original problem description.

This gap motivates a more fundamental question: *what is the minimal structural description required for an LLM to perform correct reasoning?* While natural language prompts are often verbose and rich in surface detail, it is unknown which components are functionally necessary for preserving reasoning. Rather than treating prompt reduction as an efficiency or compression problem, we frame it as a diagnostic question of *structural sufficiency*: how much of the original problem structure can be removed before functional behavior changes?

To investigate this question, we adopt code generation as a controlled testbed. Reasoning in code generation requires goal-directed inference, decomposition into coherent subroutines, and adherence to strict syntactic and semantic constraints, including control flow, variable dependencies, and data types (Li et al., 2022a). Crucially, code generation allows functional equivalence to be evaluated automatically through executable unit tests, providing an objective criterion for determining whether

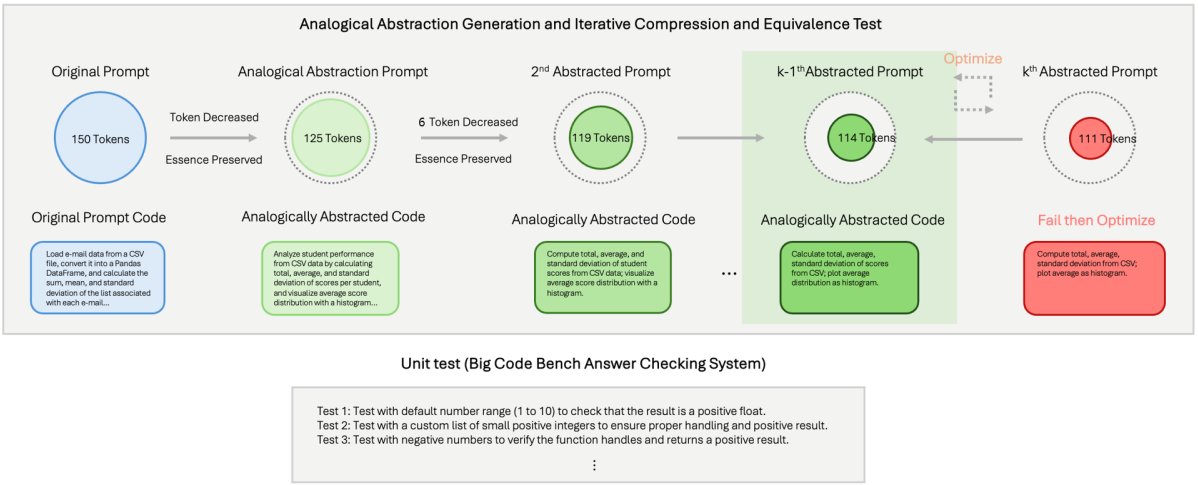

Figure 1: Analogical abstraction and compression pipeline. The process begins with the original prompt, where the LLM's generated code passes all predefined unit test cases, followed by the creation of an initial analogical abstraction that preserves the core functional essence while reducing surface detail (e.g., 125 tokens). This abstraction is then iteratively compressed (e.g., 119, 114 tokens), each time maintaining equivalence with the original task as verified by passing all functional unit tests. If a compressed prompt (e.g., 111 tokens) fails to preserve correctness, an optimization phase is triggered, interpolating between the last successful and failed versions.

reasoning has been preserved under abstraction.

Our evaluation considers only problems that a given model already solves correctly under the original prompt, as verified by passing all associated unit tests. For each such problem, the model is prompted to generate an analogical abstraction of the original description. We then replace the original prompt with this abstraction and evaluate whether the same model, without access to the original description, continues to produce correct code. Success indicates that the abstracted description is structurally sufficient to induce a reasoning process functionally equivalent to that triggered by the original prompt.

To further probe the boundary of structural sufficiency, we introduce an iterative abstraction procedure that progressively removes information from successful analogical descriptions. Starting from an abstraction that preserves correctness, the model is repeatedly prompted to produce shorter or more abstract variants, each of which is evaluated for functional equivalence (Figure 1). By identifying the transition point at which abstraction leads to failure, this procedure localizes the structural elements essential for correct reasoning. We emphasize that this process is not intended as a practical prompt compression algorithm, but as a diagnostic probe of reasoning robustness.

The contributions of this work are fourfold. First, we evaluate analogical abstraction in LLMs as a generative capacity rather than an externally supplied prompting technique. Second, we adopt functional equivalence, verified through unit test success, as a principled and executable criterion for assessing reasoning preservation. Third, we introduce an iterative abstraction and refinement procedure that serves as a diagnostic mechanism for identifying minimal structurally sufficient descriptions. Finally, we analyze how different abstraction styles and model choices exhibit distinct failure modes, revealing systematic differences in structural sensitivity across LLMs.

2 Related Works

Abstraction in Cognitive Representation. Abstraction has long been studied as a core cognitive mechanism for omitting surface features while preserving relational structure essential for reasoning. Our abstraction typology follows the distinctions discussed by Gentner and Asmuth (Gentner and Asmuth, 2019), which identify three major forms. Relational abstraction retains structural relationships across perceptually dissimilar instances, enabling category-level generalization, as in the concept of carnivore spanning cats, sharks, and spiders through the relation "X eats animals." Early transfer studies demonstrate that inducing such structural schemas enables humans to solve novel problems despite surface variation (Gick and Holyoak, 1983;

Bassok and Holyoak, 1989).

Analogical abstraction extracts shared relational schemas via structural alignment between domains (Gentner, 1983; Gentner and Markman, 1997). For example, mapping atoms to solar systems yields an abstract representation centered on a governing force and orbiting entities. Computational and cognitive models further detail how such mappings are accessed, represented, and manipulated during reasoning (Hummel and Holyoak, 1997; Gentner and Forbus, 2011). Metaphoric abstraction arises through figurative reuse, whereby concrete concepts acquire abstract interpretations while partially preserving relational structure (Jamrozik et al., 2016; Lakoff and Johnson, 1980), as when an anchor shifts from a physical object to a conceptual stabilizer. These abstraction forms align with the dual-process taxonomy proposed by Reed (Reed, 2016), which distinguishes between eliminating surface attributes and extracting generalizable relational patterns.

Analogical Reasoning in Large Language Models. Recent studies show that large language models (LLMs) exhibit non-trivial analogical reasoning capabilities, achieving strong performance on abstract pattern induction tasks such as non-visual Raven's Progressive Matrices (Webb et al., 2023). LLMs have also been shown to benefit from analogical exemplars provided prior to problem solving, outperforming zero-shot and few-shot baselines in mathematical and code generation tasks (Yasunaga et al., 2023). Other approaches construct large-scale resources that explicitly organize analogical mappings across domains, further demonstrating the expressive capacity of analogy in LLMs (Yuan et al., 2023).

However, existing work primarily evaluates analogy as an externally supplied input or augmentation strategy. Even when analogies are generated by the model itself, prior studies do not test whether such abstractions are *structurally sufficient* to replace the original problem description. In particular, they do not evaluate whether an abstraction alone, when presented to the same model without access to the original prompt, can reliably trigger a reasoning process that leads to the same functional outcome. In contrast, our work treats analogical abstraction as a generative object of study and evaluates its sufficiency through functional equivalence, measured by executable unit test success.

Prompt Compression and Structural Sufficiency. A separate line of work investigates prompt compression, aiming to reduce token redundancy through deletion, scoring, or filtering techniques such as prompt distillation and token pruning (Mu et al., 2023; Pan et al., 2024; Nagle et al., 2024; Xia et al., 2025; Shi et al., 2025). These methods typically operate at the surface level, targeting modifiers, function words, or repeated patterns, and are often evaluated in terms of efficiency or performance degradation under length constraints.

Our work differs fundamentally in objective and interpretation. Rather than optimizing prompt length or inference efficiency, we use abstraction-induced reduction as a diagnostic probe to study structural sufficiency in LLM reasoning. The iterative abstraction and refinement procedure is not proposed as a practical compression algorithm, but as a mechanism for identifying which structural elements are necessary for preserving functional behavior. Accordingly, success and failure under abstraction are interpreted as signals of structural sensitivity, not as measures of compression quality. This framing allows us to analyze when and why reasoning breaks down as structural information is progressively removed, complementing prior compression-focused studies with a structurally grounded analysis.

3 Methodology

3.1 Dataset

We used BigCodeBench-Instruct (Zhuo et al., 2024), a large-scale code generation benchmark consisting of 1,140 problems. BigCodeBench-Instruct is released under the Apache-2.0 licence and is publicly downloadable at `https://huggingface.co/bigcode/bigcodebench-instruct`. Among existing datasets such as HumanEval (Chen et al., 2021), APPS (Hendrycks et al., 2021), and CodeContests (Li et al., 2022b), BigCodeBench stands out in task diversity. Unlike datasets that focus narrowly on algorithmic solutions, BigCodeBench includes a wide range of practical coding tasks such as function orchestration, file I/O, data visualization, and interaction with libraries. Consequently, it provides a suitable testbed for analyzing structural sufficiency in LLM reasoning under realistic programming constraints. Rather than evaluating practical deployment scenarios, we use BigCodeBench-Instruct to leverage its executable

unit tests, which enable precise verification of functional equivalence under abstraction. To isolate the effect of structural abstraction, we restrict our evaluation to tasks that each model originally solved correctly using the full prompt. Our goal is not to maximize compression, but to probe whether abstracted descriptions preserve the same functional behavior, as determined by passing the exact same test cases.

3.2 Analogical Abstraction Generation

Our core assumption is as follows: instead of using an explicit prompt that fully specifies the conditions and structure of a problem, a single analogical abstraction, one that implicitly embeds the same structure, can suffice to trigger accurate reasoning in an LLM. This relies on the internalized prior knowledge of the LLM to reconstruct the problem structure and arrive at the correct solution. Crucially, the same model (denoted as M_0) must generate analogical abstraction, ensuring that the prior knowledge space is shared. To generate this compressed analogical form, we input the original problem P_0 into M_0 along with the following instruction that is designed to activate relevant prior knowledge (see Appendix A for a full prompt):

> "Transform the following programming problem using analogical abstraction while preserving all technical specifications."

This process produces a compressed prompt P_{next} from the original prompt P_0, removing surface-level details while retaining the structural elements necessary to elicit the same reasoning process. The following examples illustrate this transformation.

- **Original:** "Given a list of integers, compute the sum of absolute differences between adjacent elements for all possible permutations, then return the average of these sums."
  Analogy: "It's like running an experiment to compare the total height difference in every possible lineup."

- **Original:** "Find all combinations of numbers that sum exactly to a target value. Each number can be used only once, and duplicate combinations are not allowed."
  Analogy: "It's like picking items for a shopping list that perfectly fits a budget."

Algorithm 1 Structural Sufficiency Probing via Iterative Abstraction

Input: Original prompt P_0, model M, threshold Δ
Output: Compressed prompt P^*

$A_0 \leftarrow M(P_0)$ {Save original output}
$P_s \leftarrow P_0$; $L_s \leftarrow \text{TokenLength}(P_0)$; $P \leftarrow P_0$
while True **do**
  $P_{\text{next}} \leftarrow \text{Compress}(P)$; $A_{\text{next}} \leftarrow M(P_{\text{next}})$
  if A_{next} passes all test cases **then**
    $L_{\text{next}} \leftarrow \text{TokenLength}(P_{\text{next}})$
    if $L_s - L_{\text{next}} < \Delta$ **then**
      break
    end if
    $P_s \leftarrow P_{\text{next}}$; $L_s \leftarrow L_{\text{next}}$; $P \leftarrow P_{\text{next}}$
  else
    $P_b \leftarrow P_{\text{next}}$
    while True **do**
      $P_m \leftarrow \text{Interpolate}(P_s, P_b)$; $A_m \leftarrow M(P_m)$
      if A_m passes all test cases **then**
        $L_m \leftarrow \text{TokenLength}(P_m)$
        if $L_s - L_m \geq \Delta$ **then**
          $P_s \leftarrow P_m$; $L_s \leftarrow L_m$; $P \leftarrow P_m$
          break {Resume outer loop}
        else
          break {Compression too small}
        end if
      else
        $P_b \leftarrow P_m$ {Interpolate again}
      end if
      if stopping criterion is met **then**
        break outer loop
      end if
    end while
  end if
end while
return $P^* \leftarrow P_s$

We emphasize that this procedure is not proposed as an efficient optimization or compression algorithm. Instead, it serves as a diagnostic probe to identify the boundary at which removing structural information causes reasoning failure.

3.3 Iterative Abstraction and Equivalence Test

Building on the above examples, we formalize the analysis procedure as an *iterative abstraction*

probe (see Algorithm 1) that starts from the original prompt P_0. The first application of the abstraction function $\mathrm{Compress}(P_0)$ corresponds to generating an initial analogical abstraction. Subsequent iterations progressively remove structural detail while testing whether functional equivalence with P_0 is preserved. We emphasize that this procedure is not intended as an efficient optimization or deployment-time compression algorithm, but as a diagnostic mechanism for identifying the boundary at which removing structural information causes reasoning failure.

Iterative Abstraction Phase. Starting from P_0, a structural abstraction function $\mathrm{Compress}(P)$ is applied iteratively, allowing the LLM to generate a shorter candidate prompt P_{next}. The model output $M(P_{\text{next}})$ is then evaluated for functional equivalence with $M(P_0)$ using the same unit tests. If all test cases pass and the reduction in prompt length exceeds a minimum threshold Δ, the abstraction is accepted and becomes the new reference point P_s for further probing. This phase incrementally removes structural information while monitoring when reasoning behavior begins to degrade.

Refinement Phase. If a candidate abstraction P_{next} fails the unit tests, a refinement phase is triggered to localize the failure boundary. An interpolation function $\mathrm{Interpolate}(P_s, P_b)$ generates an intermediate prompt P_m that lies between the last successful abstraction P_s and the failed abstraction P_b in terms of structural detail. If P_m passes all test cases and satisfies the length reduction criterion, it is accepted as the new reference abstraction. Otherwise, the failed boundary is updated by setting $P_b \leftarrow P_m$, progressively narrowing the interval between P_s and P_b. This process resembles a backtracking line search, not for optimization, but for identifying the point at which essential structural cues are lost.

Stopping Criterion and Output. Abstraction progress is tracked using a token length function $\mathrm{TokenLength}(P)$. Let L_s denote the length of the last accepted abstraction and L_{next} the length of the current candidate. A candidate is accepted only if $L_s - L_{\text{next}} \geq \Delta$. The procedure terminates when no further structurally valid abstraction can be found that preserves functional equivalence. The final output P^* represents the shortest verified description that remains functionally equivalent to the original prompt P_0 under this probing process.

3.4 Experimental Setup

We conduct experiments using three large language models: GPT-4o-2024-11-20 (via OpenAI API (OpenAI, 2024)), Claude-3.5-Sonnet-2024-10-22 (via Anthropic API (Anthropic, 2024)), and Llama-3.3-70B (via TogetherAI API (TogetherAI, 2024)). All models are evaluated in zero-shot settings without additional fine-tuning. To ensure deterministic outputs, we set the temperature to 0.0 across all models. For consistency, we fixed the top_p parameter to 1.0, left max_tokens unspecified, and retained all other sampling configurations at their respective platform defaults.

For all abstraction experiments, the threshold Δ was set to 1 to maximize the sensitivity of detecting minimal structural changes. Experiments were performed locally on a single workstation equipped with an AMD Ryzen 5 5600X CPU (6 cores / 12 threads, 3.7 GHz), 32 GB RAM, an NVIDIA GeForce RTX 3060 Ti GPU (8 GB VRAM), and a 2 TB SSD, running Windows 10 Home (build 19045) and Python 3.9.32.

3.5 Evaluation

To characterize the structural sufficiency and sensitivity of LLM reasoning under abstraction, we report four diagnostic metrics: Analogical Abstraction Success Rate, Abstraction Robustness Rate, Average Final Abstracted Length, and Structural Reduction Ratio.

Token length. Token length is measured for the *problem description only* (excluding the fixed instruction template and other non-description fields) to match the definition used in Structural Reduction Ratio. For GPT-4o and Claude-3.5-Sonnet, we use the token counts reported by the provider APIs. For Llama-3.3-70B, we use the model's tokenizer provided by the inference stack. Token counts are interpreted within-model and are not used for absolute cross-model comparisons.

Analogical Abstraction Success Rate. This metric measures the proportion of tasks for which the initial analogical abstraction P_1 yields correct outputs without further abstraction. Let N_{pass} denote the number of tasks the model originally solves using the full prompt P_0, and N_{analogy} the number of those tasks for which P_1 also passes all test cases.

We define:

$$\text{Analogical Abstraction Success Rate} = \frac{N_{\text{analogy}}}{N_{\text{pass}}} \times 100. \tag{1}$$

This metric captures the model's ability to reconstruct task structure from minimal analogical cues alone.

Abstraction Robustness Rate. This metric measures the proportion of tasks for which functional equivalence is preserved under further abstraction beyond the initial analogical description. Abstraction is considered robust only if both the analogical abstraction P_1 and the final abstracted prompt P^* pass all test cases. Let N_{robust} denote the number of such tasks. We define:

$$\text{Abstraction Robustness Rate} = \frac{N_{\text{robust}}}{N_{\text{analogy}}} \times 100. \tag{2}$$

This reflects the model's tolerance to progressive removal of structural information before reasoning fails.

Average Final Abstracted Length. This metric reports the average token length of the final abstracted prompt P^*, computed over tasks for which abstraction remains functionally equivalent. It indicates how compactly a task can be represented while preserving correct reasoning behavior.

Structural Reduction Ratio. This metric quantifies the average proportion of structural description removed relative to the original prompt. Let L_0 and L^* denote the token lengths of P_0 and P^*, respectively. We define:

$$\text{Structural Reduction Ratio} = \frac{L_0 - L^*}{L_0} \times 100. \tag{3}$$

Higher values indicate that larger portions of the original description can be removed without compromising functional correctness.

Ablation Study. To analyze how different abstraction styles affect structural sufficiency, we compare analogical, metaphorical, and relational abstraction. All variants use the same abstraction probing pipeline. The evaluation follows the same protocol as the model comparison setting, using the diagnostic metrics above. In this context, the Analogical Abstraction Success Rate is referred to as the *Abstraction Success Rate*. These comparisons are intended to reveal differences in how abstraction styles preserve or discard distinct types of structural information, rather than to compare competing compression techniques.

4 Results

4.1 Model Comparison

Table 1 characterizes how three large language models differ in *structural sufficiency* under abstraction. Because each model is evaluated on the subset of tasks it originally solved using the full prompt, results are interpreted descriptively rather than as population-level performance comparisons. Bootstrap confidence intervals quantify variability without assuming paired samples or distributional normality.

Analogical abstraction success. GPT-4o shows the highest probability that a single analogical abstraction P_1 can replace the original description while preserving functional equivalence (64.86%, 95% CI [61.2, 68.4]). Claude-3.5-sonnet (33.84%, [30.1, 37.7]) and Llama-70B (47.29%, [43.5, 51.1]) require more explicit structural detail, indicating lower tolerance to abstraction at the initial stage.

Robustness under further abstraction. Among tasks where P_1 succeeds, GPT-4o remains robust under additional abstraction in 59.65% of cases ([56.0, 63.3]). Llama-70B (41.00%, [37.3, 44.9]) and Claude-3.5-sonnet (33.84%, [30.1, 37.8]) show faster degradation as structural detail is removed. These patterns suggest model-specific differences in how sharply reasoning performance collapses beyond the abstraction boundary.

Boundary location. For tasks that remain functionally correct, GPT-4o converges to the most compact verified descriptions (16.19 tokens on average, [15.7, 16.8]) and the highest Structural Reduction Ratio (45.83%, [42.1, 49.6]). The other models stabilize at longer descriptions and lower reduction ratios, indicating that their failure boundaries occur earlier along the abstraction trajectory.

4.2 Comparison by Abstraction Strategy

Table 2 compares abstraction styles on GPT-4o, isolating the effect of representational choice. Bootstrap confidence intervals indicate that observed differences are stable across resampled task sets, though not interpreted as hypothesis tests.

Success and robustness. Analogical abstraction consistently shows the highest success and robustness rates (64.86% and 59.65%), followed by

Table 1: Structural sufficiency under abstraction by model ($\Delta = 1$). Metrics are computed on the subset of tasks each model originally solved using the full prompt. Values are reported with 95% bootstrap confidence intervals (CI) obtained by resampling tasks within each model's solvable set. Structural Reduction Ratio is based on description-only reduction.

Metric	GPT-4o	Claude-3.5-sonnet	Llama-70B
Analogical Abstraction Success Rate (%)	64.86 [61.2, 68.4]	33.84 [30.1, 37.7]	47.29 [43.5, 51.1]
Abstraction Robustness Rate (%)	59.65 [56.0, 63.3]	33.84 [30.1, 37.8]	41.00 [37.3, 44.9]
Average Final Abstracted Length	16.19 [15.7, 16.8]	23.61 [22.9, 24.4]	21.84 [21.1, 22.6]
Structural Reduction Ratio (%)	45.83 [42.1, 49.6]	23.70 [20.4, 27.1]	27.25 [24.0, 30.6]

Table 2: Structural sufficiency under different abstraction styles (GPT-4o, $\Delta = 1$). Values are reported with 95% bootstrap confidence intervals (CI) over tasks solved by GPT-4o using the original prompt. Structural Reduction Ratio is based on description-only reduction.

Metric	Analogical	Metaphorical	Relational
Abstraction Success Rate (%)	64.86 [61.2, 68.4]	54.66 [50.9, 58.4]	39.91 [36.4, 43.6]
Abstraction Robustness Rate (%)	59.65 [56.0, 63.3]	45.99 [42.3, 49.8]	34.92 [31.6, 38.5]
Average Final Abstracted Length	16.19 [15.7, 16.8]	27.13 [26.2, 28.1]	16.29 [15.8, 16.9]
Structural Reduction Ratio (%)	45.83 [42.1, 49.6]	8.98 [6.4, 11.9]	49.49 [45.6, 53.3]

metaphorical abstraction, with relational abstraction being the least reliable. The ordering of abstraction styles is preserved across bootstrap samples, suggesting a robust trend in structural preservation rather than noise-driven variation.

Compactness versus reliability. Relational abstraction yields the highest Structural Reduction Ratio (49.49%, [45.6, 53.3]) and very short final descriptions, but succeeds on substantially fewer tasks. Metaphorical abstraction, by contrast, requires longer descriptions (27.13 tokens, [26.2, 28.1]) to maintain correctness, resulting in minimal reduction. These results highlight a trade-off between representational compactness and reliability, with analogical abstraction offering the most favorable balance for GPT-4o.

4.3 Case Analysis of Abstraction Outcomes

To complement aggregate statistics, Appendix B presents representative abstraction outcomes for GPT-4o. Rather than serving as anecdotal examples, these cases illustrate how specific types of structural information are preserved or lost under different abstraction styles, providing qualitative grounding for the observed trends.

The five cases are grouped by the number of abstraction styles that preserve functional equivalence. In Case 1, all three abstraction styles succeed (16.5% of tasks). Cases 2–4 involve exactly one failure among the three styles (1.6% combined). Case 5 represents the rare scenario (0.2%) in which only analogical abstraction succeeds; no task exhibits metaphor-only or relational-only success.

Failures consistently correspond to the omission or distortion of task-critical structure (e.g., numerical constraints, operation order, encoding steps), rather than superficial wording changes, reinforcing the interpretation of abstraction outcomes as indicators of structural sufficiency.

Case 1: All abstraction strategies succeed. This case illustrates tasks where the required structure is simple enough that multiple abstraction styles preserve functional intent. The original prompt requests a histogram with a fixed number of bins. Even when metaphor introduces figurative framing, the essential binning constraint remains recoverable, and the relational form preserves the schema directly.

Cases 2 to 4: Two abstraction strategies succeed. These cases reflect style-specific failure modes. In Case 2, metaphorical abstraction introduces symbolic language that obscures the underlying mathematical constraints, leading to incorrect execution. In Case 3, an analogical framing misrepresents the key operation (frequency counting in CSV data), suggesting that certain analogies can distort the operative structure even when they appear plausible. In Case 4, relational abstraction omits a critical step related to base64 encoding, consistent with under-specification failures. Across cases, failures correspond to the removal or deformation of task-critical structure rather than superficial wording changes.

Case 5: Only analogical abstraction succeeds. This rare case (0.2%) illustrates a setting where analogical abstraction preserves dependency structure sufficiently to trigger correct execution, while metaphorical and relational abstractions fail. Here, metaphorical language remains too vague to preserve numerical logic, and the relational form distorts the update and aggregation sequence required for correct list and array construction.

4.4 Discussion

Our results reveal substantial inter-model variation in tolerance to structural abstraction, consistent with prior findings on differences in abstraction and reasoning behavior (Yasunaga et al., 2023; Qu et al., 2025). GPT-4o exhibits consistently higher structural sufficiency than Claude-3.5-Sonnet and Llama-70B, both in initial abstraction success and robustness under further structural removal. Rather than reflecting general superiority, this pattern suggests differences in how models encode and retrieve structured representations during reasoning, a factor previously associated with abstraction-aligned behavior (Mitchell et al., 2023; Kim et al., 2025).

Abstraction success in this setting is not driven by removing redundant surface content, but by preserving task-critical structure that enables reconstruction of latent constraints. This distinction aligns with prior work separating structural abstraction from shallow prompt shortening (Fei et al., 2025). Across models, reasoning failure emerges abruptly once essential structural cues are removed, supporting a boundary-based view of abstraction sensitivity (Jha et al., 2024; Zheng et al., 2023).

Abstraction strategy strongly determines where this failure boundary lies. Although all strategies follow the same probing procedure, analogical abstraction preserves functional behavior across a broader range of structural reductions. Case-level analysis (Section 4.3) shows that analogical abstraction remains effective in isolation, whereas metaphorical and relational abstractions fail when structural alignment degrades. These failures are systematic, reflecting distortions or omissions of task-critical relations rather than gradual noise accumulation.

Metaphorical abstractions often obscure operational structure as abstraction progresses, with figurative framing displacing numerical or procedural constraints, consistent with prior analyses of metaphor-based prompting (Kramer, 2025). Relational abstractions, by contrast, frequently underspecify sequencing and dependency information, limiting their reliability in procedural reasoning tasks, a limitation also noted in relation-driven inference studies (Gorur et al., 2024).

Taken together, these findings support the view that analogical abstraction serves as a particularly effective structural cue for activating pretrained knowledge in LLMs. Analogical descriptions provide compact relational scaffolds that guide reconstruction of the underlying task, aligning with evidence that successful prompts rely on structural alignment rather than verbosity (Fei et al., 2025; Li et al., 2024). Methodologically, this work reframes abstraction-induced reduction not as an optimization objective but as a diagnostic lens for identifying the minimal conditions under which LLM reasoning is preserved.

5 Conclusion

This work examined whether large language models can generate *analogical abstractions* that are structurally sufficient to replace original problem descriptions while preserving functional behavior. By framing abstraction as a diagnostic probe rather than an optimization objective, we analyzed how progressively removing structural information affects reasoning outcomes.

Using executable unit tests in code generation as an objective criterion for functional equivalence, we showed that many tasks remain solvable under substantial abstraction, provided that task-critical relational structure is preserved. We observed pronounced differences across models in their tolerance to abstraction, as well as systematic differences across abstraction styles. In particular, analogical abstraction consistently preserved functional behavior more reliably than metaphorical or relational formulations, which failed due to identifiable forms of structural misalignment or underspecification.

Overall, these findings suggest that prompt length alone is not a meaningful proxy for reasoning difficulty. Instead, reasoning success depends on whether essential structural cues are preserved, and abstraction outcomes provide a useful lens for analyzing how large language models internalize and reconstruct structure during reasoning.

6 Limitations

This study has several limitations that should be considered when interpreting the results, many of which also highlight directions for future research.

First, all experiments are conducted on code generation tasks from BigCodeBench-Instruct. While this domain enables precise verification of functional equivalence via unit tests, the findings may not directly generalize to other reasoning domains such as open-ended question answering, mathematical problem solving, or commonsense reasoning, where correctness is harder to define and evaluate. Extending the abstraction probing framework to such domains will require alternative correctness signals and evaluation protocols beyond executable tests.

Second, the evaluation is restricted to within-model abstraction, requiring the same model to both generate and consume the abstracted prompt. This design isolates structural sufficiency within a single model, but does not capture how abstractions behave across models with different architectures or training regimes. Future studies on cross-model abstraction transfer may help disentangle shared structural priors from model-specific representation strategies.

Third, the abstraction procedure is explicitly designed as a diagnostic probe rather than an efficient system. We do not measure inference cost, latency, or total token usage across abstraction iterations, and the results should not be interpreted as evidence of deployment-time efficiency gains. Understanding how structurally sufficient abstractions interact with practical efficiency constraints remains an open question.

Finally, our analysis focuses exclusively on model behavior and does not examine how humans perceive or interact with abstracted prompts. Human interpretability, usability, and potential misalignment introduced by abstraction are therefore outside the scope of this work. Investigating how humans comprehend and collaborate around abstracted descriptions represents an important direction for future human–LLM research.

References

Anthropic. 2024. Introducing claude 3.5 sonnet. https://www.anthropic.com/news/claude-3-5-sonnet. Accessed 2025-08-02.

Miriam Bassok and Keith J Holyoak. 1989. Interdomain transfer between isomorphic topics in algebra and physics. *Journal of Experimental Psychology: Learning, Memory, and Cognition*, 15(1):153.

Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Ponde de Oliveira Pinto, Jared Kaplan, Harri Edwards, Yuri Burda, Nicholas Joseph, Greg Brockman, Alex Ray, Raul Puri, Gretchen Krueger, Michael Petrov, Heidy Khlaaf, Girish Sastry, Pamela Mishkin, Brooke Chan, Scott Gray, and 39 others. 2021. Evaluating large language models trained on code.

Weizhi Fei, Xueyan Niu, Guoqing Xie, Yingqing Liu, Bo Bai, and Wei Han. 2025. Efficient prompt compression with evaluator heads for long-context transformer inference. *arXiv preprint arXiv:2501.12959*.

Dedre Gentner. 1983. Structure-mapping: A theoretical framework for analogy. *Cognitive science*, 7(2):155–170.

Dedre Gentner and Jennifer Asmuth. 2019. Metaphoric extension, relational categories, and abstraction. *Language, Cognition and Neuroscience*, 34(10):1298–1307.

Dedre Gentner and Kenneth D Forbus. 2011. Computational models of analogy. *Wiley interdisciplinary reviews: cognitive science*, 2(3):266–276.

Dedre Gentner and Arthur B Markman. 1997. Structure mapping in analogy and similarity. *American psychologist*, 52(1):45.

Mary L Gick and Keith J Holyoak. 1980. Analogical problem solving. *Cognitive psychology*, 12(3):306–355.

Mary L Gick and Keith J Holyoak. 1983. Schema induction and analogical transfer. *Cognitive psychology*, 15(1):1–38.

Deniz Gorur, Antonio Rago, and Francesca Toni. 2024. Can large language models perform relation-based argument mining? *arXiv preprint arXiv:2402.11243*.

Dan Hendrycks, Steven Basart, Saurav Kadavath, Mantas Mazeika, Akul Arora, Ethan Guo, Collin Burns, Samir Puranik, Horace He, Dawn Song, and Jacob Steinhardt. 2021. Measuring coding challenge competence with apps. *NeurIPS*.

Keith J Holyoak and Paul Thagard. 1996. *Mental leaps: Analogy in creative thought*. MIT press.

John E Hummel and Keith J Holyoak. 1997. Distributed representations of structure: A theory of analogical access and mapping. *Psychological review*, 104(3):427.

Anja Jamrozik, Marguerite McQuire, Eileen R Cardillo, and Anjan Chatterjee. 2016. Metaphor: Bridging embodiment to abstraction. *Psychonomic bulletin & review*, 23(4):1080–1089.

Siddharth Jha, Lutfi Eren Erdogan, Sehoon Kim, Kurt Keutzer, and Amir Gholami. 2024. Characterizing prompt compression methods for long context inference. *arXiv preprint arXiv:2407.08892*.

Zae Myung Kim, Anand Ramachandran, Farideh Tavazoee, Joo-Kyung Kim, Oleg Rokhlenko, and Dongyeop Kang. 2025. Align to structure: Aligning large language models with structural information. *arXiv preprint arXiv:2504.03622*.

Oliver Kramer. 2025. Conceptual metaphor theory as a prompting paradigm for large language models. *arXiv preprint arXiv:2502.01901*.

George Lakoff and Mark Johnson. 1980. Metaphors we live by, university of chicago press.

Yujia Li, David Choi, Junyoung Chung, Nate Kushman, Julian Schrittwieser, Rémi Leblond, Tom Eccles, James Keeling, Felix Gimeno, Agustin Dal Lago, and 1 others. 2022a. Competition-level code generation with alphacode. *Science*, 378(6624):1092–1097.

Yujia Li, David Choi, Junyoung Chung, Nate Kushman, Julian Schrittwieser, Rémi Leblond, Tom Eccles, James Keeling, Felix Gimeno, Agustin Dal Lago, Thomas Hubert, Peter Choy, Cyprien de Masson d'Autume, Igor Babuschkin, Xinyun Chen, Po-Sen Huang, Johannes Welbl, Sven Gowal, Alexey Cherepanov, and 7 others. 2022b. Competition-level code generation with alphacode. *Science*, 378(6624):1092–1097.

Zongqian Li, Yinhong Liu, Yixuan Su, and Nigel Collier. 2024. Prompt compression for large language models: A survey. *arXiv preprint arXiv:2410.12388*.

Melanie Mitchell, Alessandro B Palmarini, and Arseny Moskvichev. 2023. Comparing humans, gpt-4, and gpt-4v on abstraction and reasoning tasks. *arXiv preprint arXiv:2311.09247*.

Jesse Mu, Xiang Li, and Noah Goodman. 2023. Learning to compress prompts with gist tokens. *Advances in Neural Information Processing Systems*, 36:19327–19352.

Alliot Nagle, Adway Girish, Marco Bondaschi, Michael Gastpar, Ashok Vardhan Makkuva, and Hyeji Kim. 2024. Fundamental limits of prompt compression: A rate-distortion framework for black-box language models. *Advances in Neural Information Processing Systems*, 37:94934–94970.

OpenAI. 2024. Hello gpt-4o. https://openai.com/index/hello-gpt-4o/. Accessed 2025-08-02.

Zhuoshi Pan, Qianhui Wu, Huiqiang Jiang, Menglin Xia, Xufang Luo, Jue Zhang, Qingwei Lin, Victor Rühle, Yuqing Yang, Chin-Yew Lin, and 1 others. 2024. Llmlingua-2: Data distillation for efficient and faithful task-agnostic prompt compression. *arXiv preprint arXiv:2403.12968*.

Yuxiao Qu, Anikait Singh, Yoonho Lee, Amrith Setlur, Ruslan Salakhutdinov, Chelsea Finn, and Aviral Kumar. 2025. Learning to discover abstractions for llm reasoning. In *ICML 2025 Workshop on Programmatic Representations for Agent Learning*.

Stephen K Reed. 2016. A taxonomic analysis of abstraction. *Perspectives on Psychological Science*, 11(6):817–837.

Junhan Shi, Yijia Zhu, Zhenning Shi, Dan Zhao, Qing Li, and Yong Jiang. 2025. Speccot: Accelerating chain-of-thought reasoning through speculative exploration. In *ES-FoMo III: 3rd Workshop on Efficient Systems for Foundation Models*.

TogetherAI. 2024. Announcing llama 3.3 70b. https://www.together.ai/blog/llama-3-3. Accessed 2025-08-02.

Taylor Webb, Keith J Holyoak, and Hongjing Lu. 2023. Emergent analogical reasoning in large language models. *Nature Human Behaviour*, 7(9):1526–1541.

Heming Xia, Chak Tou Leong, Wenjie Wang, Yongqi Li, and Wenjie Li. 2025. Tokenskip: Controllable chain-of-thought compression in llms. *arXiv preprint arXiv:2502.12067*.

Michihiro Yasunaga, Xinyun Chen, Yujia Li, Panupong Pasupat, Jure Leskovec, Percy Liang, Ed H Chi, and Denny Zhou. 2023. Large language models as analogical reasoners. *arXiv preprint arXiv:2310.01714*.

Siyu Yuan, Jiangjie Chen, Changzhi Sun, Jiaqing Liang, Yanghua Xiao, and Deqing Yang. 2023. Analogykb: Unlocking analogical reasoning of language models with a million-scale knowledge base. *arXiv preprint arXiv:2305.05994*.

Huaixiu Steven Zheng, Swaroop Mishra, Xinyun Chen, Heng-Tze Cheng, Ed H Chi, Quoc V Le, and Denny Zhou. 2023. Take a step back: Evoking reasoning via abstraction in large language models. *arXiv preprint arXiv:2310.06117*.

Terry Yue Zhuo, Minh Chien Vu, Jenny Chim, Han Hu, Wenhao Yu, Ratnadira Widyasari, Imam Nur Bani Yusuf, Haolan Zhan, Junda He, Indraneil Paul, and 1 others. 2024. Bigcodebench: Benchmarking code generation with diverse function calls and complex instructions. *arXiv preprint arXiv:2406.15877*.

Appendix A: Prompts Used for LLMs

Analogy Transformation Prompt

Transform the following programming problem using analogical abstraction while preserving ALL technical specifications.

Original problem: `{original_instruction}`

ANALYSIS-BASED PRESERVATION RULES:

100% REQUIRED (Must appear in ALL analogy transformations): - "The function should output with:" specification (100% of BigCodeBench problems) - "You should write..." code template (100% of BigCodeBench problems) - "def task_func" function signature (100% of BigCodeBench problems)

CONDITIONAL PRESERVATION (Only if present in original problem): - "Args:" specifications (only 0.5% of problems, but must preserve if present) - "The function should raise the exception for:" (26% of problems) - "Notes:" or "Note that" sections (5% of problems)

WHAT YOU MUST NEVER CHANGE: - Any "The function should output with:" sections - Any "You should write..." code templates - Any function signatures or import statements - Any Args, Exception, or Notes specifications if they exist - Any technical specifications

EXAMPLE TRANSFORMATION: Original: "Calculates the average of the sums of absolute differences between each pair of consecutive numbers for all permutations of a given list."

Analogy: "This is like averaging the total unbalance from all possible random lineups of people by height."

[All technical specifications remain IDENTICAL]

CRITICAL: Only transform the conceptual description. All technical specifications must be copied exactly as written.

Description Compression Prompt

Compress the programming problem DESCRIPTION ONLY using analogical abstraction to be as short as possible while maintaining essential algorithmic information.

Original description: `{description}`

COMPRESSION RULES: 1. Keep core algorithmic concepts 2. Preserve key mathematical operations 3. Maintain logical flow and relationships 4. Remove redundant words and verbose explanations 5. Use concise, technical language

FORBIDDEN: Do NOT add technical specifications, function signatures, or example formats. ONLY compress the problem description.

Example: Original: "Calculate the average of the sums of absolute differences between each pair of consecutive numbers for all permutations of a given list"

Compressed: "This is like averaging the total unbalance from all possible random lineups of people by height."

Refined Abstraction Prompt

Your latest abstraction went too far and caused test failure. Find the RIGHT BALANCE between your previous attempts.

Previous SUCCESSFUL description: `{previous_successful_description}`

Latest FAILED description: `{failed_abstract_description}`

Test failure reason: `{test_failure_message}`

TARGET BALANCE: - MORE compressed than successful version - LESS compressed than failed version - Preserve enough technical detail for correct code generation

LENGTH GUIDANCE: Target `{target_min}` to `{target_max}` words

EXAMPLE PROGRESSION: Original: "Calculate sum of squared values for all elements in array" Successful: "Compute total of squared array elements" Failed: "Like adding energy" Better Balance: "Calculate total squared values from collection"

Code Generation Prompt Example

Solve the following programming problem.

Problem description: `{instruction}`

Code template: `{code_prompt}`

CRITICAL REQUIREMENTS: 1. Function name MUST be 'task_func' (absolutely unchanged) 2. Function signature and parameters MUST match exactly 3. Parameter types and default values MUST be preserved 4. Return type MUST match the specification 5. Import statements must be written before defining the function 6. Write only code without comments or explanations 7. Write completely functional code that passes all tests

FORBIDDEN: Changing function name, modifying parameters/types/defaults, changing return type, adding/removing parameters

Response format:
“‘python import library def task_func(parameters): return result “‘

Appendix B: Abstraction Case Examples

Case 1 – All Abstractions Succeed (16.5%)

Original	Create a histogram of the specified attribute from a list of objects and return the histogram plot. Constants: `NUM_BINS` (30).
Analogical	Generate a histogram of rainfall frequencies across defined ranges.
Metaphorical	Whispering chests seek order; sort them into 30 bins to unveil their treasure map.
Relational	Generate a histogram of an entity property with `NUM_BINS` (default 30).

Case 2 – Only Metaphorical Abstraction Fails (0.8%)

Original	Generates all possible combinations of the provided numbers … and sums the logarithms.
Analogical	Sum logarithms of harmonic products across all note subsets.
Metaphorical	Alchemists distill elements into essence (product) and wisdom (logarithmic sum) through mystical fusion.
Relational	Sum logarithms of products across all combinations.

Case 3 – Only Analogical Abstraction Fails (0.2%)

Original	Reads a CSV file, counts word frequencies, and returns $(\text{word}, \text{count})$ pairs sorted in descending order.
Analogical	Sort ingredients by frequency across recipes in descending order.
Metaphorical	Navigate a fruit bazaar in CSV form, tally sales, and crown the most popular, using the stall map as your guide.
Relational	Sort frequencies from CSV by delimiter.

Case 4 – Only Relational Abstraction Fails (0.6%)

Original	Add a current timestamp to a Python dictionary, serialize to JSON, then base64-encode (ASCII).
Analogical	Timestamp, format, and encrypt a diary entry.
Metaphorical	Capture time's essence, script it in JSON's language, and veil it in base64's enigma for safekeeping.
Relational	Base64-encode JSON profile with timestamp.

Case 5 – Only Analogical Abstraction Succeeds (0.2%)

Original	Append a random integer [0, 100] to `my_list` and return a NumPy array of random floats whose size equals the sum of the updated list.
Analogical	Add a random book to a library, then generate random pages summing to the total pages of all books.
Metaphorical	A sprite guards a gem, echoing the chest's value through a numerical dance in array form.
Relational	Create a NumPy array of random floats sized by the sum of a list after appending a random integer (0–100).