An Approach for Systematic Decomposition of Complex LLM Tasks

Anonymous ACL submission

Abstract

Large Language Models (LLMs) suffer from reliability issues on complex tasks, as existing decomposition methods are heuristic and rely on agent or manual decomposition. This work introduces a novel, systematic decomposition framework that we call `Analysis of CONstraint-Induced Complexity` (`ACONIC`), which models the task as a constraint problem and leverages formal complexity measures to guide decomposition. On combinatorial (SAT-Bench) and LLM database querying tasks (Spider), we find that by decomposing the tasks following the measure of complexity, agent can perform considerably better.

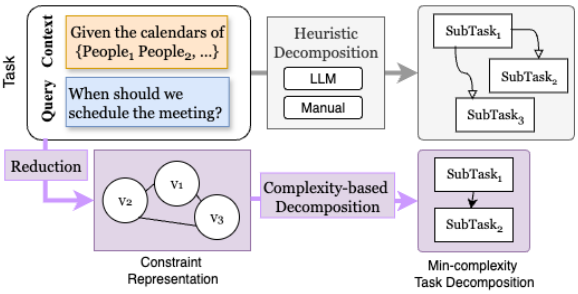


Figure 1: Given a scheduling task, existing approaches use heuristic LLM or manual task decomposition methods (top row). Our framework reduces the task into a constraint satisfaction problem that allows for a systematic decomposition that minimizes the problem complexity (bottom row).

1 Introduction

Large Language Models (LLMs) have demonstrated impressive competence across a wide range of reasoning, programming, and problem-solving tasks. Yet, when faced with *complex tasks* that require deep multi-step reasoning or combinatorial search, even state-of-the-art models often fail to produce correct results in a single forward pass.

A growing body of work addresses this limitation through *task decomposition*. Instead of solving a task monolithically, these methods break it into smaller, more tractable subtasks. One popular line of work, starting from chain-of-thought (Wei et al., 2022), is to use LLMs for decomposition (Yao et al., 2023; Khot et al., 2023; Pourreza and Rafiei, 2023; Chen et al., 2024). Other methods rely on domain experts to decompose the task into workflows and provide access to tools that shoulder parts of the task (Wang et al., 2024; Singh et al., 2024). We summarize these methods as the grey path in Figure 1.

While decomposition seeks to break more complex tasks into workflows of simpler subtasks, existing approaches are largely heuristic. When is a task "complex"? How should it be decomposed? A principled measure of task complexity would enable systematic decomposition strategies and the ability to study tasks of comparable difficulty, and provide guidance on when tools are needed.

In this paper, we introduce a formal complexity framework for LLM tasks by reducing them into *constraint satisfaction problems* (specifically, 3-SAT). We use properties of the induced constraint graph (graph size and treewidth) as measures of task complexity. Building on these measures, we propose a decomposition method based on (Bodlaender, 1998) that minimizes subtask complexity under this formalization, yielding decompositions that preserve global satisfiability while maximizing local solvability. Along the purple path in Figure 1, we first reduce the task—modeled as a context that describes a set of constraints and a query that must reason over the constraints—into a formal constraint satisfiability problem. We then decompose the constraint problem and construct a workflow over the subtasks defined for each subproblem.

We use SAT-BENCH (Wei et al., 2025) and the NL2SQL SPIDER (Yu et al., 2019) benchmark to study this framework. We find that the task measures define **frontiers of difficulty** that separate tasks easily solvable by LLMs from those that

are nearly impossible without structural assistance. In addition, complexity-guided decomposition improves completion on SATBench and outperforms Tree-of-Thoughts on Spider by 3–8 accuracy points across difficulty levels.

2 Method

At a high level, we find that a class of agent tasks that admit constraint formulations can be reduced to constraint satisfaction problems (CSPs), such that solving the CSP is equivalent to solving the original task. This formulation makes explicit the dependency structure among task variables, which allows us to leverage tree decomposition from constraint processing theory to decompose complex tasks into smaller, weakly coupled subproblems that can be reasoned about locally and composed consistently.

2.1 Reduction to Constraint Satisfaction

We model a class of agent tasks as instances of constraint satisfaction problems (CSPs). A CSP is defined by a tuple $\mathcal{C} = \langle X, D, \mathcal{R} \rangle$, where X is a finite set of variables, D assigns each variable a finite domain, and $\mathcal{R}$ is a set of constraints over subsets of variables.

Within this framework, the *state* of the agent is a (partial) assignment to variables in X, and an *action* corresponds to proposing or updating values for a subset of variables, or acquiring information that refines the constraints. The agent's objective is to construct a complete assignment that satisfies all constraints in $\mathcal{R}$.

We illustrate this abstraction using a simple meeting scheduling example.

> *Motivation Example — Meeting Scheduling.*
> Alice needs to meet with Bob and Charlie separately and asks an agent to schedule both meetings. The system records show:
>
> - Alice: `["Alice is available at morning in office r", "Alice is available in the afternoon at office t"]`
> - Bob: `["Bob is available in the afternoon at office t"]`
> - Charlie: `["Charlie is available at morning in office r", "Charlie is available in the afternoon at office t"]`
>
> **Task.** How should the agent compose the invitation emails to Alice–Bob and Alice–Charlie?

In this example, resolving conflicts is non-trivial because assignments for one meeting constrain the feasible choices for the other. For instance, choosing a time for Alice and Charlie may invalidate the only feasible option for Alice and Bob.

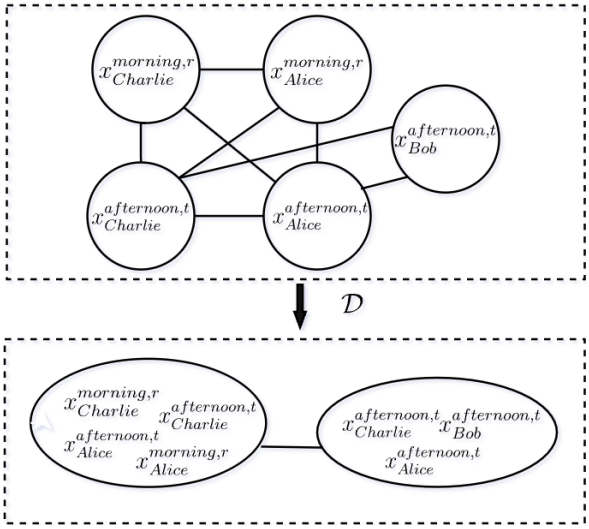


Figure 2: CSP graph (and its tree decomposition) for Meeting Scheduling example. Constructed based on CSP representation in appendix E

We introduce Boolean variables $x_p^{(t,\ell)}$ indicating whether person p attends location ℓ at time t. Let $\mathcal{P}$ denote the set of participants, $\mathcal{T}$ time slots, and $\mathcal{L}$ locations. For each person p, let $\mathcal{A}_p \subseteq \mathcal{T} \times \mathcal{L}$ denote feasible slots, and for each pair (p, p') let $\mathcal{F}_{p,p'} = \mathcal{A}_p \cap \mathcal{A}_{p'}$.

The goal is to select exactly one feasible meeting slot for each desired pair $\mathcal{F}_{A,B}$ and $\mathcal{F}_{A,C}$, and to ensure that no one attend two meetings simultaneously. This leads to a set of constraints over the variables $\{x_p^{(t,\ell)}\}$ (As demonstrated in appendix E).

The agent must produce a valid initiation of the variables X that satisfies all the constraints. Our objective is to find a problem decomposition strategy that provide the agent with observations over a subset of the problem.

2.2 Tree Decomposition for Agent Tasks

As illustrated in 2, the CSP induces a constraint graph $G = (X, E)$, where variables are nodes and an edge connects two variables if they co-occur in some constraint. It provides a structural abstraction of the task, capturing which task variables are mutually dependent through constraints. This representation allows us to reason about task structure independently of concrete values, and motivates decomposition strategies that group tightly coupled variables into localized subproblems. We therefore explore how tree decomposition (Bodlaender,

1998) of this graph can be used to organize and simplify agent reasoning.

Tree decomposition maps G into a tree of bags $\mathcal{D} = (T, \{B_i\})$, where each bag $B_i \subseteq X$ satisfies:

- Every constraint is contained in some bag.
- For any variable x, the set of bags containing x forms a connected subtree.

Following these properties, tree decomposition ensures that if each bag enforces local consistency on its induced constraints, then these local solutions can be merged into a global satisfying assignment as long as shared variables agree along the tree.

Among all tree decompositions $\mathcal{D} = (T, \{B_i\})$ of G, there exists an optimal decomposition $\mathcal{D}^* \in \arg\min_{\mathcal{D}} \max_i |B_i|$. The corresponding minimum possible maximum bag size, $b^*(G) := \min_{\mathcal{D}} \max_i |B_i|$, induces the *treewidth* of G as $\mathrm{tw}(G) = b^*(G) - 1$. We use this quantity to characterize the intrinsic structural complexity of the underlying constraint problem: smaller $\mathrm{tw}(G)$ implies that the task can be decomposed into smaller, more weakly-coupled subproblems.

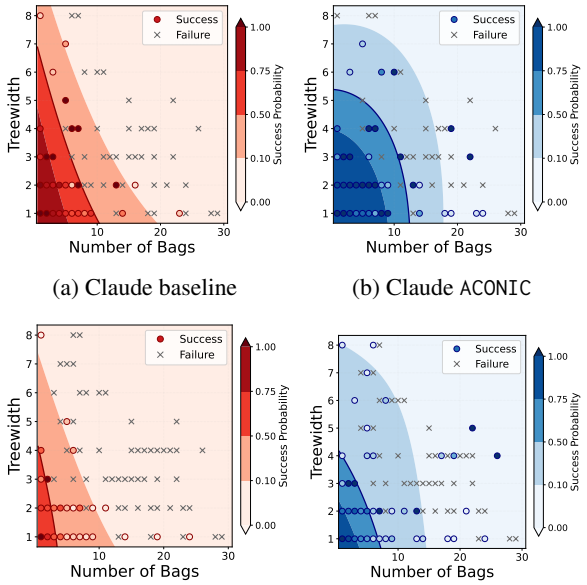
We exploit this structure to decompose reasoning into minimal, locally consistent subtasks. Each bag corresponds to a subproblem over a small variable set. The agent observes and reasons following the subproblems, while keeping consistency over intersections and ensuring global satisfiability. This leads to a principled decomposition strategy whose complexity is governed by the intrinsic treewidth of the task.

3 Experiments

We now evaluate our framework against chain-of-thought decomposition (Wei et al., 2022) and Tree of Thoughts (Yao et al., 2023) on two datasets: SAT-based story problems (SAT-bench) and natural language to SQL (Spider).

3.1 SAT-Bench

SAT-Bench (Wei et al., 2025) uses (i) an underlying SAT problem to construct (ii) a natural-language story describing the same constraints, and (iii) an alignment from story entities to their SAT representations. Instead of asking the LLM for an immediate satisfiability judgment, we prompt it to produce value assignments step-by-step to ensure a complete understanding of the assignment process. In this process, the agent's state is a partial assignment



(a) Claude baseline (b) Claude ACONIC

(c) LLaMA baseline (d) LLaMA ACONIC

Figure 3: Our complexity measures define **frontiers of difficulty**. SOTA models like Claude shifts the boundaries towards the right, while ACONIC's decomposition consistently pushes the frontiers towards more complex problems.

to SAT variables, and progress corresponds to extending this assignment while maintaining consistency with the underlying constraints. Concretely, we treat the SAT-Bench CNF as a Boolean CSP, where the goal is to find an assignment that satisfies all clauses.

At each round of variable assignment, the agent is provided with the story, the variable to SAT mapping rules, and a set of observations. We tested agent with two different configurations of observations. In the chain-of-thought baseline, the agent observes all condition observations. In the tree-decomposition setup, the agent instead observes all conditions belonging to the same subproblem induced by the decomposition.

We evaluated Llama-3-70B on all tasks and Claude3.5-Sonnet on half the tasks (randomly sampled). Figure 3 plots the success/failure of each task by its complexity measures, and exhibits frontiers of difficulty beyond which tasks are too difficult or too simple. For the baseline, there appears to be a fixed "total task complexity" evidenced by the trade-off between problem treewidth and number of bags (left column). In contrast, ACONIC decomposition pushes the frontier outward and is able to successfully complete more complex tasks. Overall, ACONIC increases the task completion rate from $49.3\% \rightarrow 58.1\%$ using Claude, and

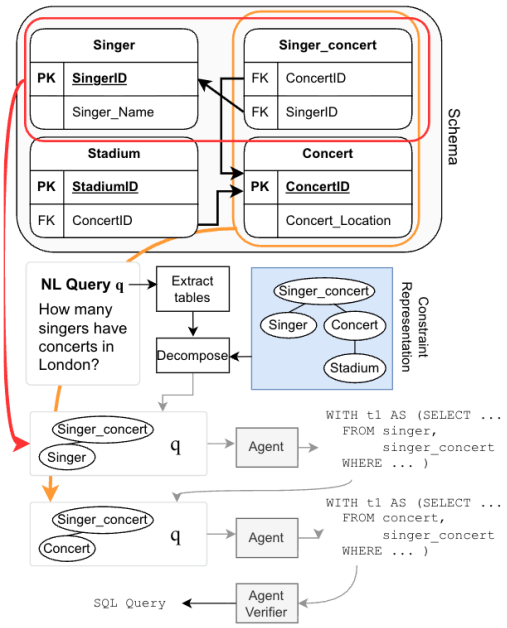


Figure 4: Instead of giving the full database schema and NL query q to an agent, ACONIC decomposes the NL task into two subtasks, each given a subset of the database schema, q, and the output of the previous subtask if any.

$21.5\% \rightarrow 36.5\%$ using LLaMA—a $9-15\%$ improvement on both models.

3.2 Natural Language to SQL

Natural language to SQL (NL2SQL) tasks translate a natural language query over a database into a desired SQL query. A major challenge is constructing the join condition to connect tables referenced in the query. For example, the schema in Figure 4 contains 4 tables about singers and concerts. Although the NL query only refers to singers and concerts, the LLM needs to infer that `singer_in_concert` is needed for a valid join path.

The complexities of both the query (the number of tables referenced) and the database schema (number of tables and their foreign key graph) affect the agent's ability to construct queries with the appropriate join graphs. In addition, database theory (Gottlob et al., 2001) already models a database as a constraint graph (tables are nodes, and foreign keys are edges) and a query as a subgraph; this representation has also been used for NL2SQL approaches (Wang et al., 2019). Thus, we use NL2SQL to study decomposition.

We use the popular Spider NL2SQL benchmark dataset (Yu et al., 2019). It contains hundreds of databases, each containing up to 37 tables, 90 foreign keys, and potentially over 100 columns.

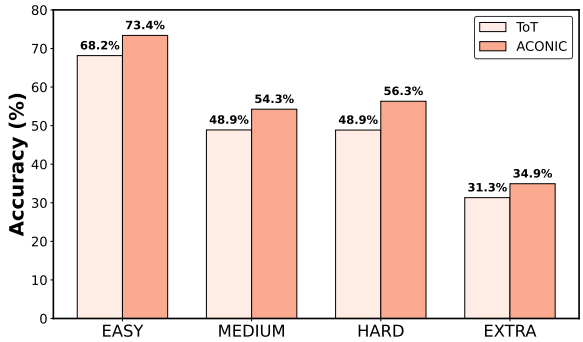


Figure 5: Accuracy performance of Tree of Thought (ToT) and ACONIC (Ours) on Spider EASY, MEDIUM, HARD, and EXTRA tasks

Each task consists of a database schema S, the natural language (NL) query q, and a ground-truth SQL query q^*. The original benchmark submits (S, q) to the LLM, which can result in incorrect or invalid join conditions. Given the tables referenced in q, ACONIC decomposes the schema into subgraphs with minimized maximal complexities and constructs a workflow (grey arrows). For instance, the first subtask gives the agent q and the schemas for subgraph `singer-singer_concert` (red arrow), and asks it to construct the appropriate `WITH` clause. The next subtask does the same with the second subgraph (orange arrow), but is also provided the output from the previous subtask. A verification agent cleans up minor errors to build the final SQL query.

4 Conclusions

ACONIC is a principled framework for decomposing complex LLM tasks by reducing them to constraint satisfaction problems and quantifying complexity via treewidth. Unlike heuristic methods, ACONIC minimizes local complexity while preserving global satisfiability. Across SATBench and Spider, this yielded **frontiers of task difficulty** that define LLM reasoning limits. Complexity-based decomposition improved completion rates by up to 15% on SAT-bench. On Spider datasets, the accuracy was increased over ToT by up to 7.5 points (and +5.4 points overall). These results suggest a path towards theoretically grounded, reliable multi-step LLM systems. Figure 5 reports Spider accuracy for ACONIC and Tree-of-Thoughts (ToT). Across different difficulty levels, ACONIC consistently outperforms ToT by 3.6–7.5 accuracy points, with an overall gain of +5.4 points ($50.68\% \rightarrow 56.09\%$).

5 Limitations

ACONIC provides a principled framework to analyze and reduce task complexity, but it does not yet constitute a fully autonomous decomposition or reasoning system. It is also not intended to be a complete solution for general task decomposition, but rather as a theoretical and empirical examination of how *constraint-induced complexity* affects the problem-solving ability of LLMs.

Our focus has been on evaluating how complexity-guided decomposition impacts performance relative to heuristic baselines such as chain-of-thought prompting, rather than on direct comparison with other theoretically grounded or learning-based decomposition frameworks. For this reason, we also do not claim that our final decomposition system is superior to existing solutions to the benchmark tasks. For instance, SAT-bench can be directly solved using a constraint solver. Similarly, the NL2SQL tasks can be readily solved using simple path-finding algorithms over the database schema.

The tasks that we evaluated can be conveniently modeled as constraint satisfiability problems. Although many other practical problems, such as deadlock detection in databases, resource scheduling in operating systems, task placement in distributed systems, and control/data-flow construction in agentic programming, can be formulated as constraint satisfiability problems, they often cannot be logically represented completely, either due to question ambiguity, lack of transparency of the agent actions, or fuzzy contextual information. In these cases, future work might study hybrid decomposition approaches that mix logical and commonsense constraints in a task.

References

Hans L Bodlaender. 1998. A partial k-arboretum of graphs with bounded treewidth. *Theoretical Computer Science*, 209(1-2):1–45.

Zhiruo Chen, Simeng Qiu, Victor Zhong, and Percy Liang. 2024. Divide-and-conquer prompting for complex reasoning. *arXiv preprint arXiv:2402.05359*.

Georg Gottlob, Nicola Leone, and Francesco Scarcello. 2001. Hypertree decompositions: A survey. In *International Symposium on Mathematical Foundations of Computer Science*, pages 37–57. Springer.

Tushar Khot, Harsh Trivedi, Matthew Finlayson, Yao Fu, Kyle Richardson, Peter Clark, and Ashish Sabharwal. 2023. Decomposed prompting: A modular approach for solving complex tasks. *arXiv preprint arXiv:2306.04179*.

Mohammadreza Pourreza and Davood Rafiei. 2023. Din-sql: Decomposed in-context learning of text-to-sql with self-correction. *Advances in Neural Information Processing Systems*, 36:36339–36348.

Amanpreet Singh, Chenglei Si, Kurt Shuster, Jing Xu, and Jason Weston. 2024. Agent–computer interfaces: An approach for task-grounded agent evaluation. In *NeurIPS*.

The Terminal-Bench Team. 2025. Terminal-bench: A benchmark for ai agents in terminal environments.

Bailin Wang, Richard Shin, Xiaodong Liu, Oleksandr Polozov, and Matthew Richardson. 2019. Rat-sql: Relation-aware schema encoding and linking for text-to-sql parsers. *arXiv preprint arXiv:1911.04942*.

Xingyao Wang, Yangyi Chen, Lifan Yuan, Yizhe Zhang, Yunzhu Li, Hao Peng, and Heng Ji. 2024. Executable code actions elicit better llm agents. *arXiv preprint arXiv:2402.01030*.

Anjiang Wei, Yuheng Wu, Yingjia Wan, Tarun Suresh, Huanmi Tan, Zhanke Zhou, Sanmi Koyejo, Ke Wang, and Alex Aiken. 2025. Satbench: Benchmarking llms' logical reasoning via automated puzzle generation from sat formulas. *Preprint*, arXiv:2505.14615.

Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Brian Ichter, Fei Xia, Ed Chi, Quoc Le, and Denny Zhou. 2022. Chain-of-thought prompting elicits reasoning in large language models. *arXiv preprint arXiv:2201.11903*.

Shunyu Yao, Dian Yu, Jeffrey Zhao, Izhak Shafran, Tom Griffiths, Pengfei Cao, Karthik Narasimhan, and Yingyu Song. 2023. Tree of thoughts: Deliberate problem solving with large language models. *arXiv preprint arXiv:2305.10601*.

Tao Yu, Rui Zhang, Kai Yang, Michihiro Yasunaga, Dongxu Wang, Zifan Li, James Ma, Irene Li, Qingning Yao, Shanelle Roman, Zilin Zhang, and Dragomir Radev. 2019. Spider: A large-scale human-labeled dataset for complex and cross-domain semantic parsing and text-to-sql task. *Preprint*, arXiv:1809.08887.

A Appendix: SAT-Bench Transformation Details

In this appendix, we provide a detailed explanation of how we perform the formal reduction and experimental setup for the SAT-Bench benchmark.

A.1 Notation and Inputs

A SAT-Bench instance provides:

- A CNF over Boolean variables $V = \{v_1, \ldots, v_n\}$ with clauses $\Phi = \{C_1, \ldots, C_m\}$, where each clause with variable $\mathrm{vars}(C_j) = \{i_{j,1}, \ldots, i_{j,k_j}\}$ and $C_j = \left(\bigvee_{r=1}^{k_j} \ell_{j,r}\right)$ and each literal $\ell_{j,r} \in \{v_i, \neg v_i\}$.
- A natural-language story Q and an alignment map ψ from story entities/conditions to SAT literals or short CNF fragments (as specified by SAT-Bench).

A.2 CSP Encoding of SAT Clauses

We build a CSP instance $\mathcal{R} = \langle X, D, C \rangle$ as follows. $X = \{x_1, \ldots, x_n\}$ (one variable per SAT variable v_i) with domain of each variable being $D_{x_i} = \{0, 1\}$

For each clause $C_j = \left(\bigvee_{r=1}^{k_j} \ell_{j,r}\right)$, let $\mathrm{vars}(C_j) = \{i_{j,1}, \ldots, i_{j,k_j}\}$ be the indices of variables appearing in C_j (ignoring polarity). Define the *literal evaluation* map for an assignment $\mathbf{t}_j$ of all possible combinations in the domain D ($\mathbf{t}_j \in D^{\mathrm{vars}(C_j)}$) to be with all local assignments to variables $vars(C_j)$ so that clause C_j is satisfied. Therefore the constraint over the variables would be $\mathbf{T}_j$ with all valid $\mathbf{t}_j$. Each $\mathbf{T}_j$ thus defines an $|\mathrm{vars}(C_j)|$-ary constraint relation over the variables appearing in C_j.

We encode the variable V to CSP variables with initiation of 0 (False) and the clauses $C_j = \left(\bigvee_{r=1}^{k_j} \ell_{j,r}\right)$ with literal $\ell_{j,r} \in \{v_r, \neg v_r\}$ as clause-wise constraint $\mathbf{T}_j$ including all the literals in C_j.

A.3 Tree Decomposition on the Induced CSP Graph

We represent each CSP variable as a node in the graph. For every constraint (clause) C_j defined over the variable set $\mathrm{vars}(C_j)$, we connect all variables in $\mathrm{vars}(C_j)$ with edges, thereby forming a clique of size $|\mathrm{vars}(C_j)|$. The resulting undirected graph captures the variable–constraint dependencies of the SAT instance.

To analyze structural complexity, we perform *tree decomposition* on this graph, which partitions the variable set into overlapping subgraphs (bags) while minimizing the size of the largest bag. The minimum achievable largest bag size minus one corresponds to the graph's *treewidth*, which we use as a key measure of problem complexity and locality in constraint reasoning (Bodlaender, 1998) in chart 6a, 6b, 6c, 3d

In our implementation, we compute the tree decomposition using `SageMath`, which provides efficient heuristics (e.g., minimum fill-in) and exact solvers for treewidth minimization.

A.4 Experiment Workflow

Algorithm 1 SAT-Bench Workflow

```
Input: NL story Q, CNF clauses Φ, mapping ψ
Build CSP:
    X ← variables in Φ
for each clause C in Φ do
        T_C ← all satisfying assignments of variables in C
end for
Build constraint graph:
for each clause C in Φ do
        connect variables in C in graph
end for
Tree decomposition:
    Bags ← tree_decompose(graph)
Agent interaction:
    State ← empty assignment
while not solved and budget not exceeded do
    if CoT / ToT / DaC then
            Obs ← (current violation, all clauses in Φ)
    else if ACONIC then
            Violated ← clauses violated under State
            Bag ← bag containing variables of Violated
            Obs ← (current violation, clauses inside Bag)
    end if
    Action ← LLM(State, Q, ψ, Obs)
    State ← update assignment(Action)
end while
Return final assignment
```

We detail here the experimental setup for both the baseline *Chain-of-Thought* (CoT) decomposition and the proposed `ACONIC` method. The

agent interacts with the environment over multiple rounds, progressively constructing a variable assignment.

At each round, the agent is provided with:

- the natural-language query Q,
- the variable-to-literal mapping ψ,
- the current violation status (i.e., whether any clause in Φ is unsatisfied under the current partial assignment), and
- a set of **observations**, which differ (and is the only difference) between the baseline and `ACONIC` configurations.

Baseline with full observation. In the Chain-of-Thought baseline, the agent receives the complete observation set, including all conditions (clauses) in the instance. This corresponds to a global reasoning setup in which the agent has full visibility of the constraint space.

`ACONIC` (Decomposed Observation). In the `ACONIC` configuration, the agent receives only the subset of conditions relevant to its current focus. Specifically, when a clause is violated, the system identifies the *bag* in the tree decomposition that contains the variables of the violated clause, and provides the agent with all conditions (clauses) associated with that bag.

Agents are required to provide a set of value assignment to the variables according to the observations, violation status, query, and the mapping ψ. The process ends until all the variables are properly set such that no violation is caused, or a maximum interaction budget is reached.

A.5 Implementation

We implement SAT-Bench experiment as a Terminal-Bench task (Team, 2025) to ensure reproducibility and comparability between agents.

B Appendix: Spider database decomposition details

B.1 Notation and Inputs

The Spider benchmark (Yu et al., 2019) provides a large-scale collection of natural language (NL) queries paired with their corresponding database schemas, gold-standard SQL queries, and example data. In our experiments, we use only the NL query and the associated schema as inputs to the framework, the gold SQL and data are reserved for evaluation.

For queries involving multiple tables, we model the join operations as a *constraint satisfaction problem* (CSP) defined over the Cartesian product of the participating tables. Each attribute join condition introduces a binary constraint between the respective table variables, and the feasible assignments correspond to tuples that jointly satisfy all join and filter predicates.

B.2 CSP Encoding of Tables and Joins

For the NL-to-SQL task, we model relational queries as constraint satisfaction problems (CSPs) defined over database tables. Each table in the schema is represented as a CSP variable, and its domain corresponds to the power set of all possible tuple combinations within that table:

$$X_i = \text{Table}_i, \qquad D_i = 2^{\text{Tuples}(\text{Table}_i)}.$$

Hence, a value assignment to X_i specifies a subset of rows (tuples) selected from that table under the query conditions.

For a join operation between two tables T_a and T_b, the feasible results can be expressed as a subset of the Cartesian product of their tuple domains:

$$D_{a,b} \subseteq D_a \times D_b,$$

where each pair of tuples (t_a, t_b) is retained if and only if it satisfies the join predicate (e.g., matching primary–foreign key attributes, SQL defined join rules).

In this formulation, the NL query is modeled as a set of *semantic pairwise constraints* over tables and attributes. Each join, filter, or projection condition introduces a logical constraint between variable assignments. Unary constraints are applied to represent filtering conditions on individual tables (e.g., `WHERE` clauses). Thus, solving the CSP corresponds to finding a consistent set of table subsets that jointly satisfy all query semantics implied by the NL input.

B.3 Tree Decomposition on the Induced CSP Graph

As the core structural analysis step in `ACONIC`, we perform tree decomposition on the CSP graph induced by the relational constraints. The procedure is analogous to that used for the SAT-Bench experiments.

We construct a constraint graph where each node corresponds to a table variable, and an undirected

edge is added between two nodes if there is a Foreign Key constraint represented in the schema.

We then apply the tree decomposition algorithm that minimizes the maximum bag size to obtain a decomposition with minimal treewidth. The resulting tree structure defines a set of subproblems (bags), each representing a locally consistent subset of tables and constraints. As in the SAT-Bench setup, we use the `SageMath` package to compute tree decompositions, leveraging its built-in heuristics (e.g., minimum fill-in) and exact solvers for treewidth minimization.

B.4 Experiment Workflow

We compare the baseline *Tree-of-Thought* (ToT) heuristic decomposition method with the proposed `ACONIC` framework on the NL2SQL task. In both settings, the agent interacts with the environment over multiple rounds, constructing the final SQL query.

Algorithm 2 Spider Workflow

Input: NL query NLQ, database schema S
Build CSP:
  $X \leftarrow$ tables in S
Build constraint graph (foreign keys only):
for each foreign key constraint **do**
    connect corresponding tables in graph
end for
Tree decomposition:
  Bags $\leftarrow$ tree_decompose(graph)
Agent interaction:
  CTEs $\leftarrow$ empty list
while unresolved tables remain **do**
  if ToT **then**
    Obs $\leftarrow$ (current SQL, full schema S)
  else if ACONIC **then**
    Target $\leftarrow$ tables referenced in NLQ
    Bag $\leftarrow$ bag containing unresolved Target tables
    Obs $\leftarrow$ (CTEs, schema restricted to Bag)
  end if
  Fragment $\leftarrow$ LLM(NLQ, Obs, Schema S)
  append Fragment to CTEs
end while
LLM merging:
  SQL $\leftarrow$ LLM_based_merge_and_resolve(CTEs)
Return final SQL

At each round, the agent is provided with:

- the natural-language query (NLQ),
- the database schema (whose visibility differs between the baseline and `ACONIC` setups), and
- the current intermediate assignment or generated SQL fragment.

Baseline (Full Schema). In the ToT baseline, the agent receives the entire database schema at once and is asked to generate the complete SQL query that satisfies the NLQ. This configuration represents global reasoning over the full relational structure, without any decomposition or locality constraints.

`ACONIC` (Decomposed Schema). In the `ACONIC` setup, the agent begins by identifying the *target table(s)* relevant to the NLQ. This step allows `ACONIC` to eliminate irrelevant tables from the constraint graph, thereby reducing search space and improving reasoning focus. Given the current query context, the system retrieves the corresponding *bag* in the tree decomposition that contains the undetermined tables. The agent is then provided with the local schema of this bag and is asked to generate a *Common Table Expression (CTE)* that retrieves the subset of data relevant to the NLQ from these tables.

In subsequent rounds, the agent is prompted again with:

- the original NLQ,
- the current decomposed schema (limited to the active bag and its boundary variables), and
- the CTEs generated from previous rounds.

This iterative process continues until all relevant tables have been resolved.

Finally, a *verification agent* aggregates and refines the partial results, merging the generated CTEs, resolving dependencies, and applying any remaining filters or ordering conditions to produce the final executable SQL statement.

B.5 Implementation

We implement Spider experiment as a Terminal-Bench (Team, 2025) task to ensure reproducibility and comparability between agents.

C Additional Analysis: Cost Evaluation

To assess more different methods and the token costs, we randomly selected tasks from SAT-Bench covering the joint distribution of treewidth and bag count induced by tree decomposition and tested with Chain of Thought (CoT), Divide and Conquer (DaC), Tree of Thoughts (ToT), and ACONIC(Ours). Specifically, we uniformly sample problem instances across discrete bins of $(\mathrm{tw}(G), |T|)$, where $\mathrm{tw}(G)$ denotes the treewidth of the constraint graph and $|T|$ the number of bags in the corresponding decomposition. This ensures that the evaluation set is not biased toward either very simple or very complex structural regimes.

Table 1 reports the aggregate accuracy and average token usage of each method under this sampling scheme.

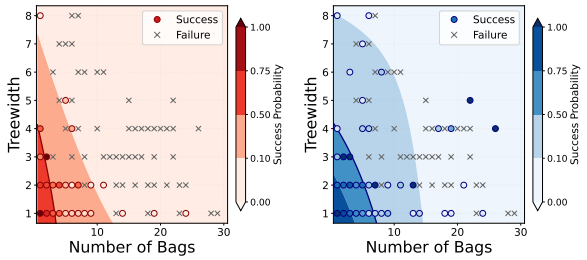
Method	Accuracy	Avg. Tokens
CoT	21.95%	24,179
DaC	21.95%	129,201
ToT	26.82%	209,976
ACONIC	**29.27%**	**19,247**

Table 1: Performance under random structural coverage across all treewidth and bag-count distributions.

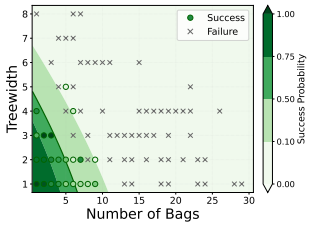
ACONIC achieves the highest accuracy while using fewer tokens than all baselines. This suggests that explicitly exploiting structural decompositions yields both effectiveness and efficiency benefits when reasoning over constraint-based tasks.

We also observe that Divide-and-Conquer (DaC) performs comparably to chain-of-thought in accuracy but incurs substantially higher token cost. This behavior is consistent with the nature of SAT and Spider tasks, which require maintaining and propagating multiple dependent variable assignments across steps. In such settings, naively decomposing the task into independent subquestions can lead to excessive context expansion without a corresponding gain in correctness.

Overall, these results support the view that structure-aware decomposition guided by the underlying constraint graph is better aligned with tasks characterized by strong variable dependencies, whereas generic divide-and-conquer strategies are less effective when subproblems are not weakly coupled.



(a) Llama Chain of Thoughts (b) Llama Aconic ACONIC



(c) Llama Tree of Thoughts

Figure 6: Task difficulty frontiers for LLaMA under CoT, ACONIC, and ToT on SAT-Bench. Each subplot shows success rate distribution based on problem complexity.

D Additional Analysis: Tree of Thoughts

We include results for Tree of Thoughts (ToT) on SAT-Bench (See figure 6. Under our ToT configuration, ToT solves 34.8% of tasks, compared to CoT's 21.5% and ACONIC's 36.5%.

Method	CoT	ToT	ACONIC
Success Rate	21.5%	34.8%	36.5%

Table 2: SAT-Bench success rate comparison across reasoning methods.

We follow the implementation in the original paper (Yao et al., 2023). The search procedure is parameterized by: (i) maximum depth T=5, i.e., the search tree expands for at most 5 levels; (ii) branching factor b=3, i.e., each node proposes 3 candidate thoughts; and (iii) frontier size k=5, i.e., we keep the top 5 candidates at each depth as the frontier for further expansion.

E Boolean Encoding and SAT-to-CSP Reduction (Meeting Scheduling Example)

We provide the mechanical steps that instantiate the meeting scheduling example as a Boolean formula and then as a CSP.

Variables and domains. Let $\mathcal{P}$ denote the set of participants, $\mathcal{T}$ the set of time slots, and $\mathcal{L}$ the

set of locations. For each person $p \in \mathcal{P}$ and slot $(t, \ell) \in \mathcal{T} \times \mathcal{L}$, introduce a Boolean variable

$$x_p^{(t,\ell)} \in \{0, 1\},$$

where $x_p^{(t,\ell)} = 1$ indicates that p attends slot (t, ℓ). For each person p, let $\mathcal{A}_p \subseteq \mathcal{T} \times \mathcal{L}$ be the feasible availability pairs extracted from the system record. For each desired meeting pair (p, p'), define the pairwise feasible set

$$\mathcal{F}_{p,p'} := \mathcal{A}_p \cap \mathcal{A}_{p'}.$$

Boolean formulation of constraints. Define meeting indicators for a pair (p, p'):

$$M_{p,p'}^{(t,\ell)} := x_p^{(t,\ell)} \wedge x_{p'}^{(t,\ell)}.$$

The task constraints consist of: (i) *Exactly-one feasible meeting slot* for each desired pair, and (ii) No simultaneous meetings for a person. Concretely,

$$\bigwedge_{(p,p') \in \{(A,B),(A,C)\}} \mathrm{EO}\Big(\{M_{p,p'}^{(t,\ell)} \mid (t, \ell) \in \mathcal{F}_{p,p'}\}\Big) \tag{1}$$

$$\wedge \bigwedge_{(t,\ell)} \neg\Big(M_{A,B}^{(t,\ell)} \wedge M_{A,C}^{(t,\ell)}\Big), \tag{2}$$

where EO denotes *Exactly One*.

SAT (CNF) representation. Let $\Phi = \{C_1, \ldots, C_m\}$ denote the resulting CNF clauses, where each clause is a disjunction of literals over Boolean atoms (e.g., $x_p^{(t,\ell)}$ or $\neg x_p^{(t,\ell)}$). The meeting scheduling instance is satisfiable iff there exists an assignment to all involved Boolean atoms that satisfies every clause in Φ.

Reduction to CSP. Given Φ, we construct a CSP $\mathcal{R} = \langle X, D, \mathcal{C} \rangle$ as follows.

Variables. X contains one CSP variable for each Boolean atom appearing in Φ. For notational simplicity, we reuse the same symbol for the CSP variable (e.g., $x_p^{(t,\ell)}$).

Domains. Each variable is Boolean with domain

$$D_x = \{0, 1\}.$$

Constraints. For each CNF clause C_j with variable set $\mathrm{vars}(C_j) \subseteq X$, we introduce $\mathbf{T}_j$ where $\mathbf{T}_j$ contains exactly those local assignments to $\mathrm{vars}(C_j)$ that satisfy clause C_j. Letting $\mathcal{C} = \{\mathbf{T}_1, \ldots, \mathbf{T}_m\}$, an assignment satisfies the CSP iff it satisfies all clause relations.

Equivalence. By construction, any satisfying assignment of the CNF formula Φ induces a satisfying assignment of $\mathcal{C}$, and any satisfying assignment of $\mathcal{C}$ satisfies every clause in Φ. Therefore, solving the resulting CSP is equivalent to solving the original meeting scheduling instance.