2024 IEEE/ACM 46th International Conference on Software Engineering: New Ideas and Emerging Results (ICSE-NIER)

ITG: Trace Generation via Iterative Interaction between LLM Query and Trace Checking

Weilin Luo
School of Computer Science and Engineering, Sun Yat-sen University
Guangzhou, China
luowlin5@mail.sysu.edu.cn

Weiyuan Fang
School of Computer Science and Engineering, Sun Yat-sen University
Guangzhou, China
fangwy3@mail2.sysu.edu.cn

Junming Qiu
School of Computer Science and Engineering, Sun Yat-sen University
Guangzhou, China
qiujm9@mail2.sysu.edu.cn

Hai Wan*
School of Computer Science and Engineering, Sun Yat-sen University
Guangzhou, China
wanhai@mail.sysu.edu.cn

Yanan Liu
School of Computer Science and Engineering, Sun Yat-sen University
Guangzhou, China
liuyn56@mail2.sysu.edu.cn

Rongzhen Ye
School of Computer Science and Engineering, Sun Yat-sen University
Guangzhou, China
yerzh@mail2.sysu.edu.cn

ABSTRACT

Due to the complexity of linear temporal logic (LTL) trace generation (PSPACE-Complete), existing neural network-based approaches will fail as the formula sizes increase. Recently, large language models (LLMs) have demonstrated remarkable reasoning capabilities, benefiting from efficient training on hyper-scale data. Inspired by this, we propose an iterative interaction framework for applying LLMs, exemplified by ChatGPT, to generate a trace satisfying a given LTL formula. The key insight behind it is to transfer the powerful reasoning capabilities of LLM to LTL trace generation via iterative interaction between LLM reasoning and logical reasoning. Preliminary results show that compared with the state-of-the-art approach, the accuracy is relatively improved by 9.7%-23.4%. Besides, we show that our framework is able to produce heuristics for new tasks, which provides a reference for other reasoning-heavy tasks requiring heuristics.

CCS CONCEPTS

• **Theory of computation** → **Modal and temporal logics**; • **Computing methodologies** → **Neural networks**.

KEYWORDS

large language model, linear temporal logic, satisfiability checking, trace generation, trace checking

ACM Reference Format:
Weilin Luo, Weiyuan Fang, Junming Qiu, Hai Wan, Yanan Liu, and Rongzhen Ye. 2024. ITG: Trace Generation via Iterative Interaction between LLM Query and Trace Checking . In *New Ideas and Emerging Results (ICSE-NIER'24), April 14–20, 2024, Lisbon, Portugal.* ACM, New York, NY, USA, 5 pages. https://doi.org/10.1145/3639476.3639779

*corresponding author.

ICSE-NIER'24, April 14–20, 2024, Lisbon, Portugal

ACM ISBN 979-8-4007-0500-7/24/04
https://doi.org/10.1145/3639476.3639779

1 INTRODUCTION

Linear temporal logic (LTL) satisfiability checking aims to answer whether a given LTL formula is satisfiable or unsatisfiable. It is a basic theoretical problem of LTL, and its problem complexity is in PSPACE-Complete [23]. Therefore, efficient LTL satisfiability checking can improve the efficiency of the LTL-SAT-heavy tasks, *e.g.*, specification repair [1, 2] and goal-conflict analysis [5, 15]. Recently, well-designed neural networks [14, 16] have achieved excellent performance and demonstrate promising potential in polynomial time. Although neural network-based approaches have significant advantages in time cost, they leave hidden dangers that are unverifiable, *i.e.*, the satisfiability results answered by neural networks cannot be verified to be true. It is the main barrier to its application in safety-critical systems. LTL trace generation aims to generate a trace to satisfy the formula, which provides verifiability of satisfiable results [16]. Hahn et al. [10] demonstrated the potential of neural networks for trace generation in polynomial time, which maintains the performance advantages in time cost of neural networks. However, we empirically discover that their work only applies to toy example, *i.e.*, the formula with small sizes.

The core challenge of LTL trace generation is that, as the number of variables and formula sizes increases, the search space of satisfiable traces can explode exponentially. It is necessary to design a heuristic to guide the search direction to alleviate the search space explosion. The existing heuristics for LTL trace generation are either manually programmed [11–13] or automatically learned [10]. The former has the limitation that it is time-consuming and its quality relies on human intelligence, while the latter has the limitation of high training overhead. Revolutionary advancements in artificial general intelligence, especially large language models (LLMs), represented by ChatGPT [18], provide a turning point. LLMs perform commendably in reasoning tasks, *e.g.*, deduction reasoning [21], code generation [6, 8, 24] and LTL translating [4, 9], which lays the groundwork for generating traces.

To this end, we propose an iterative interactive framework (ITG) between logical reasoning and LLM reasoning to guide LLMs to heuristically search for satisfiable traces. The key insight behind

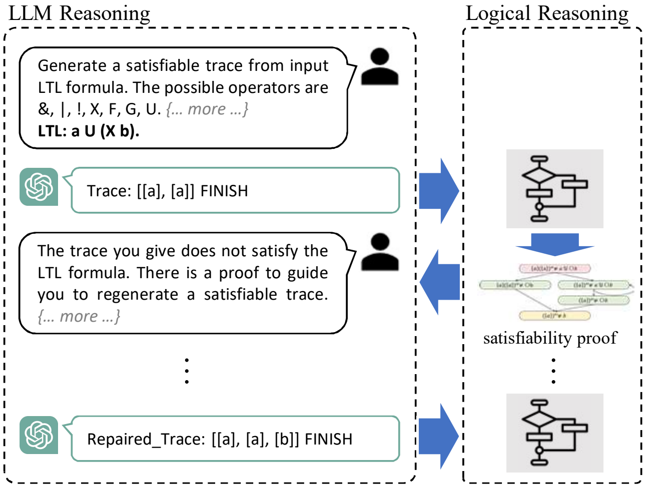

Figure 1: A overview of ITG.

it is to transfer the powerful reasoning capabilities of LLM to LTL trace generation. To induce LLMs to generate satisfiable traces, we introduce the satisfiability proof to construct prompts. Intuitively, a satisfiability proof characterizes why a trace satisfies or does not satisfy an LTL formula. It is able to prompt LLMs on how to construct satisfiable traces step by step or to provide error feedback for LLMs. ITG is divided into two parts: LLM query and trace checking. The two parts are invoked iteratively to form a loop until a satisfying trace is generated, as shown in Figure 1.

We conducted preliminary experiments on synthetic datasets[1]. We discover that the state-of-the-art (SOTA) learned heuristic [10] fails, *i.e.*, approaching the results of random guessing, as the formula sizes increases. Compared with the learned heuristic, the accuracy of ITG is relatively improved by 9.7%-23.4%, reaching SOTA performance. Besides, we evaluate the various frameworks of ITG and show that ITG is able to combine the inherent knowledge of LLMs to produce heuristics for new tasks, which provides a reference for other reasoning-heavy tasks requiring heuristics.

2 BACKGROUND

The syntax of LTL for a finite set of atomic propositions $\mathbb{P}$ includes the fundamental operators *disjunction* ($\vee$), *negation* ($\neg$), *next* ($\bigcirc$), and *until* ($\mathcal{U}$), defined as follows: $\phi := p \mid \neg\phi' \mid \phi' \vee \phi'' \mid \bigcirc\phi \mid \phi' \,\mathcal{U}\, \phi''$, where $p \in \mathbb{P} \cup \{\top\}$ and ϕ', ϕ'' are LTL formulae. For brevity, we only consider the above fundamental operators in this paper. $|\phi|$ represents the *size* of the formula ϕ, *i.e.*, the number of operators and propositions in ϕ (repeatable). The *sub-formula* of ϕ is denoted by $\mathsf{sub}(\phi)$.

LTL formulae are interpreted over *traces* of propositional states. A trace is represented in the form $\pi = s_0 \ldots (s_k \ldots s_m)^\omega$, where $s_t \in 2^{\mathbb{P}}$ is a set at time t and $(s_k \ldots s_m)^\omega$ is a *loop* indicating that $s_k \ldots s_m$ appears in order and infinitely. For every state s_i of π and every $p \in \mathbb{P}$, p holds if $p \in s_i$ or $\neg p$ holds otherwise. π_t denotes the *sub-trace* of π beginning from the state s_t, particularly, $\pi = \pi_1$. The *satisfaction relation* $\models$ is defined as follows:

$$
\begin{array}{lll}
\pi_t \models p & \text{iff} & p \in s_t, \\
\pi_t \models \neg\phi' & \text{iff} & \pi_t \not\models \phi', \\
\pi_t \models \phi' \vee \phi'' & \text{iff} & \pi_t \models \phi' \text{ or } \pi_t \models \phi'', \\
\pi_t \models \bigcirc\phi' & \text{iff} & \pi_{t+1} \models \phi' \\
\pi_t \models \phi' \,\mathcal{U}\, \phi'' & \text{iff} & \exists k \geq t \text{ s.t. } \pi_k \models \phi'' \text{ and} \\
 & & \forall t \leq j < k, \pi_j \models \phi',
\end{array}
$$

where π is a trace, ϕ', ϕ'' are LTL formulae, and $p \in \mathbb{P} \cup \{\top\}$. $\models$ and $\not\models$ are opposite. We define function $\mathsf{opp}(\models) = \not\models$ and $\mathsf{opp}(\not\models) = \models$.

3 SATISFIABILITY PROOF

To bridging logical reasoning and LLM reasoning, we define the *trace-formula relation* and the *satisfiability proof.*

Definition 3.1. Let ϕ be an LTL formula, π a trace, and r be a satisfaction relation. A *trace-formula relation* is a 3-tuple (π, r, ϕ).

Definition 3.2. Let ϕ be an LTL formula, π a trace, r be a satisfaction relation, and $\phi\ r\ \pi$ holds. The *satisfiability proof* Π of $\phi\ r\ \pi$ is a graph (V, E), where V is a set of vertices and $E \subset V \times V$ is a set of edges. A vertex $v \in V$ is a trace-formula relation. An edge $e = \langle u, v \rangle$ is directed from u to v, where v is a *support* of u. The *root* vertex is (π, r, ϕ). If $\phi_i \in \mathbb{P} \cup \{\top\}$, (π_t, r, ϕ_i) is *leaf* vertex, where $\phi_i \in \mathsf{sub}(\phi)$. V and E are initialized as $\{(\pi, r, \phi)\}$ and $\emptyset$, respectively. For each vertex $(\pi_t, r, \phi_i) \in V$, Π is computed as follows.

1. If $\phi_i = \neg\phi_j$, then $V = V \cup \{(\pi_t, \mathsf{opp}(r), \phi_j)\}$ and $E = E \cup \{\langle(\pi_t, r, \phi_i), (\pi_t, \mathsf{opp}(r), \phi_j)\rangle\}$.
2. If $\phi_i = \phi_j \vee \phi_k$ and r is $\models$, then if $\pi_t \models \phi_j$ holds, then $V = V \cup \{(\pi_t, \models, \phi_j)\}$ and $E = E \cup \{\langle(\pi_t, r, \phi_i), (\pi_t, \models, \phi_j)\rangle\}$; otherwise, $V = V \cup \{(\pi_t, \models, \phi_k)\}$ and $E = E \cup \{\langle(\pi_t, r, \phi_i), (\pi_t, \models, \phi_k)\rangle\}$.
3. If $\phi_i = \phi_j \vee \phi_k$ and r is $\not\models$, then $V = V \cup \{(\pi_t, \not\models, \phi_j), (\pi_t, \not\models, \phi_k)\}$ and $E = E \cup \{\langle(\pi_t, \not\models, \phi_i), (\pi_t, \not\models, \phi_j)\rangle, \langle(\pi_t, \not\models, \phi_i), (\pi_t, \not\models, \phi_k)\rangle\}$.
4. If $\phi_i = \bigcirc\phi_j$, then $V = V \cup \{(\pi_{t+1}, r, \phi_j)\}$ and $E = E \cup \{\langle(\pi_t, r, \phi_i), (\pi_{t+1}, r, \phi_j)\rangle\}$.
5. If $\phi_i = \phi_j \,\mathcal{U}\, \phi_k$ and r is $\models$, then if $\pi_t \models \phi_k$ holds, then $V = V \cup \{(\pi_t, \models, \phi_k)\}$ and $E = E \cup \{\langle(\pi_t, \models, \phi_i), (\pi_t, \models, \phi_k)\rangle\}$; otherwise, $V = V \cup \{(\pi_t, \models, \phi_j), (\pi_{t+1}, \models, \phi_i)\}$ and $E = E \cup \{\langle(\pi_t, \models, \phi_i), (\pi_t, \models, \phi_j)\rangle, \langle(\pi_t, \models, \phi_i), (\pi_{t+1}, \models, \phi_i)\rangle\}$.
6. If $\phi_i = \phi_j \,\mathcal{U}\, \phi_k$ and r is $\not\models$, then if $\pi_t \not\models \phi_j$ and $\pi_t \not\models \phi_k$ holds, then $V = V \cup \{(\pi_t, \not\models, \phi_j), (\pi_t, \not\models, \phi_k)\}$ and $E = E \cup \{\langle(\pi_t, \not\models, \phi_i), (\pi_t, \not\models, \phi_j)\rangle, \langle(\pi_t, \not\models, \phi_i), (\pi_t, \not\models, \phi_k)\rangle\}$; otherwise, $V = V \cup \{(\pi_t, \not\models, \phi_k), (\pi_{t+1}, \not\models, \phi_i)\}$ and $E = E \cup \{\langle(\pi_t, \not\models, \phi_i), (\pi_t, \not\models, \phi_k)\rangle, \langle(\pi_t, \not\models, \phi_i), (\pi_{t+1}, \not\models, \phi_i)\rangle\}$.

Π is a *satisfiable* satisfiability proof if the satisfaction relation of the root vertex is $\models$ and an *unsatisfiable* one otherwise.

Intuitively, a satisfiability proof explicitly reasons why a trace satisfies (the satisfiable one) or does not satisfy (the unsatisfiable one) a formula step by step based on the semantics of LTL. Figure 2 illustrates a satisfiable satisfiability proof and an unsatisfiable one.

4 ITERATIVE INTERACTIVE FRAMEWORK

In this section, we first introduce the two parts of ITG, and then give an instance to demonstrate a workflow for iterative interactions.

4.1 LLM query

In the LLM query part, we expect that LLMs can leverage rich LTL expertise to generate satisfiable traces heuristically. To this end,

[1] Our code and datasets are publicly available at https://github.com/sysulic/ITG.

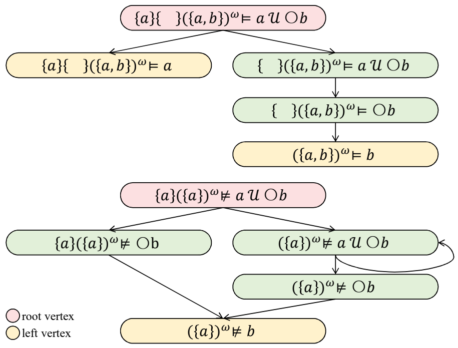

Figure 2: An example of satisfiability proof. The top is a satisfiable one and the bottom is an unsatisfiable one. Traces are interpreted with a closed-world assumption: any proposition that is not in a state is assumed not to hold in the state.

```
1 Generate a satisfiable trace from input LTL formula. The possible atomic propositions
2 will be given. The possible operators are &, |, !, X, F, G, U. Trace should be less than
3 10 states. Each state should not contain dublicated atomic propositions. The output trace
4 should be a list of states. Do not use ... in output. For example: LTL: a U (X b) Trace:
5 [[a],[],[b]] FINISH LTL: F(a & X b) Trace: [[a],[b]] FINISH LTL: G(a | ! b) Trace:
6 [[],[a]] FINISH LTL: a U (X b) Trace: [[a],[a]] FINISH
```

Figure 3: An example of an initialization prompt, including the inputs ('LTL:...') and outputs ('Trace:...') of trace generation (line 1-2), constraints (line 2-4), and the few-shot examples (line 4-6).

```
1 The trace you give does not satisfy the LTL formula. There is a proof to guide you to
2 regenerate a satisfiable trace. For example: LTL: a U (X b) Trace: [[a],[a]] Proof:{[[a]]
3 not satisfies b; [[a]] not satisfies X b; [[a]] not satisfies a U (X b); [[a],[a]] not satisfies
4 X b} Repaired_Trace: [[a], [a], [b]] FINISH
```

Figure 4: An example of an repair prompt, including the inputs ('LTL:...', 'Trace:...', and 'Proof:...') and outputs ('Repaired_Trace:...') of updating traces (line 1-2) and one-shot example (line 2-3).

we design prompts to control LLMs. Through prompts, we can effectively situate LLMs within a specific domain, thereby eliciting expertise in that domain. Although it is prevalent to control LLMs using prompts [19–21, 25, 27], our contribution is to design iterative interactive prompts for LTL trace generation.

The prompts of the LLM query include two types: initialization and repair. For the initialization prompt, we need to describe LTL trace generation, including inputs, outputs, and constraints. Besides, we adopt few-shot prompting to provide LLMs with examples of trace generation according to an LTL formula. We simulate an extremely efficient trace generation process using a satisfiable satisfiability proof. We expect that the efficient trace generation process can give LLMs enough inspiration so that LLMs are able to deal with larger and more complex formulae. Note that few-sample prompting has another potential role, that is, constraining LLMs to generate formal outputs, which provides a guarantee for the interaction of LLM reasoning and logical reasoning. Figure 3 shows an example of an initialization prompt. For the repair prompt, we also apply few-shot prompting to provide LLMs with examples of updating traces. The repair prompt is designed based on an unsatisfiable satisfiability proof generated by the trace checking part, introduced in Section 4.2. Figure 4 shows an example of a repair prompt.

4.2 Trace Checking

Given an LTL formula and a trace, we can obtain the satisfiability proof in polynomial time by modifying the trace checking algorithm [17], so we omit details of the algorithm.

4.3 Overview of ITG

Given an LTL formula, ITG (Algorithm 1) performs iterative interaction between the LLM query part (LLM reasoning) and the trace checking part (logical reasoning). LLMQUERY(ϕ, $prompt_{init}$) (*resp.* LLMQUERY(ϕ, $prompt_{repair}$)) returns the answer produced by LLMs under the initialization prompt $prompt_{init}$ (*resp.* repair prompt $prompt_{repair}$). PROOFGENERATE(π_0, ϕ, r) returns the satisfiability proof of π r ϕ. TRACECHECK(π, ϕ), whose implementation comes from the work [17], returns whether $\pi \models \phi$ holds. We set the maximum number of iterations to 4.

Algorithm 1: ITG

Input: an LTL formula ϕ.
Output: a trace π.

- $prompt_{init} \leftarrow$ generate the initialization prompt of ϕ
- $ans \leftarrow$ LLMQUERY(ϕ, $prompt_{init}$)
- **while** *not reach the maximum number of iterations* **do**
  - $\pi \leftarrow$ extract a trace based on patterns from ans
  - **if** *TRACECHECK(π, ϕ) is false* **then**
    - $\Pi \leftarrow$ PROOFGENERATE(π, ϕ, $\not\models$)
    - $prompt_{repair} \leftarrow$ generate the repair prompt based on Π
    - $ans \leftarrow$ LLMQUERY(ϕ, $prompt_{repair}$)
  - **else**
    - **return** π
- **return** *UNKNOWN*

5 PRELIMINARY EXPERIMENT

Dataset. Following the work [10], we use randltl [7] to generate random formulae, denoted by *SPOT*. The number of different atomic propositions is 8. We generate satisfiable formulae and divide them into 6 sets according to their sizes: [5, 20), [20, 40), [40, 60), [60, 80), and [80, 100). For all datasets, we randomly choose 80K formulae as a training set, 1K formulae as a validation set, and 1K formulae as a test set. We use nuXmv [3] to generate a satisfiable trace.

Evaluation metrics. We use *semantic accuracy* to evaluate the approaches. If the generated trace satisfies the formula, then it is semantically accurate. Semantic accuracy is the percentage where the generated traces are semantically accurate.

Setups. We train neural network-based approaches using the Adam optimizer and use grid search to find optimal hyperparameters. We use GPT4 [19] as the LLM query engine of ITG. We randomly generate a trace in right syntactic, denoted by random.

5.1 Are the SOTA neural network-based approach efficient?

`Transformer` [10] is the SOTA neural network-based approach in generating traces. We train and test `Transformer` on *SPOT* with different formula sizes.

Table 1: Semantic accuracy (%) of `Transformer`.

	[5, 20)	[20, 40)	[40, 60)	[60, 80)	[80, 100)
Transformer	93.30	76.21	58.66	58.32	56.35

Summary. Table 1 reports the evaluation results of `Transformer` on datasets with various formula sizes. It can be observed that `Transformer` exhibits performance degradation in tandem with the increase of formula sizes. This implies that the SOTA neural network-based approach fails to generalize in the reasoning scenarios with larger formula sizes.

5.2 What is the performance of ITG?

We compare `ITG` with `Transformer` and a variant of `ITG` (denoted by `ITG-init`). In `ITG-init`, we only use the initialization prompt to query LLMs, *i.e.*, there is no logical reasoning. We only train `Transformer` on *SPOT*-[5, 20) and test all approaches on larger formulae to evaluate the generalization ability across formula sizes.

Table 2: Semantic accuracy (%) of approaches on datasets with various formula sizes, where boldface numbers refer to the better results.

	[5, 20)	[20, 40)	[40, 60)	[60, 80)	[80, 100)
random	54.20	52.10	53.10	53.10	54.00
Transformer	**93.30**	71.30	60.30	54.20	51.60
ITG-init	69.90	61.50	59.60	57.60	57.20
ITG	91.20	**81.00**	**70.30**	**76.65**	**76.07**

Summary. Table 2 reports the evaluation results on datasets with various formula sizes. It can be seen that both `ITG-init` and `ITG` significantly outperform `random` on all datasets. These results reveal that LLMs are capable of providing heuristics for trace generation. Additionally, it is evident that `ITG` outperforms `Transformer` by a significant margin on datasets with formula sizes ≥ 20, and the performance degradation of `ITG` is relatively gentle. It confirms the generalization ability of `ITG`. Furthermore, we can observe that `ITG` obtains significant performance gains over `ITG-init` on all datasets. This implies that the proposed iterative interaction is effective in improving the trace generation performance.

5.3 Does ITG provide heuristics?

We compare `ITG` with a variant of `random` (denoted by `random-ite`). It iteratively and randomly generates traces until a satisfying trace is found or the maximum number of iterations is reached.

Summary. Figure 5 shows the changes in the average semantic accuracy on all datasets as the maximum number of iterations increases. As the maximum number of iterations increases, the advantage of `ITG` gradually becomes larger, which implies that `ITG` is able to produce heuristics that are beneficial to trace generation.

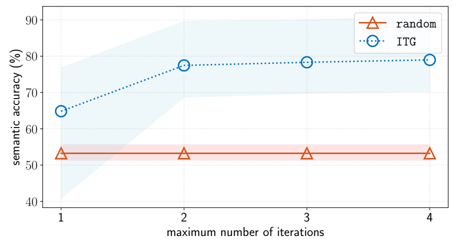

Figure 5: Average semantic accuracy on all datasets.

6 DISCUSSION AND FUTURE WORK

There is a token limit (within 8192[2]) for querying using ChatGPT's API. This results in `ITG` using ChatGPT as the query engine unable to support large-scale formulae, such as the data reported in the work [16]. In addition, since the query history is also counted within the token limit, the token limit also affects the maximum number of iterations. A larger maximum number of iterations can trigger more updates to the generated trace and trial and error opportunities, which is beneficial to improving the accuracy of `ITG`. We will experiment with LLMs without token limit in future work to support larger scale formulae.

Another threat is the query time of LLMs. Since ChatGPT currently cannot be invoked locally, ChatGPT queries need to access the service through the Internet. It will encounter potential congestion and instability of the internet, resulting in low query efficiency. We can use LLMs that can be deployed locally, *e.g.*, GLM [28] and BLOOM [22], to mitigate this threat. We leave it to future work.

As the formula sizes increase, the number of tokens corresponding to prompts will also increase. LLMs need to face reasoning challenges with long texts. The advances in the reasoning capabilities of LLMs on long texts are expected to offset this threat [26].

7 CONCLUSION

In this paper, we have empirically discovered that the SOTA neural network-based approach to trace generation fails on large-scale formulae. We have designed a framework `ITG` that iteratively interacts with LLM reasoning and logical reasoning. Preliminary results show that `ITG` outperforms the SOTA approach on synthetic datasets. Moreover, it confirms that `ITG` has heuristics for trace generation.

8 ACKNOWLEDGMENTS

This paper was supported by National Natural Science Foundation of China (No. 62276284, 61976232, 51978675), Guangdong Basic and Applied Basic Research Foundation (No. 2023A1515011470, 2022A1515011355), Guangzhou Science and Technology Project (No. 202201011699), Guizhou Provincial Science and Technology Projects (No. 2022-259), Humanities and Social Science Research Project of Ministry of Education (No. 18YJCZH006), as well as the Fundamental Research Funds for the Central Universities, Sun Yat-sen University (No. 23ptpy31).

[2]https://platform.openai.com/docs/models/gpt-4

REFERENCES

[1] Matías Brizzio, Maxime Cordy, Mike Papadakis, César Sánchez, Nazareno Aguirre, and Renzo Degiovanni. 2023. Automated Repair of Unrealisable LTL Specifications Guided by Model Counting. In *GECCO*. 1499–1507.

[2] Luiz Carvalho, Renzo Degiovanni, Matías Brizzio, Maxime Cordy, Nazareno Aguirre, Yves Le Traon, and Mike Papadakis. 2023. ACoRe: Automated Goal-Conflict Resolution. In *FASE*, Vol. 13991. 3–25.

[3] Roberto Cavada, Alessandro Cimatti, Michele Dorigatti, Alberto Griggio, Alessandro Mariotti, Andrea Micheli, Sergio Mover, Marco Roveri, and Stefano Tonetta. 2014. The nuXmv symbolic model checker. In *CAV*. 334–342.

[4] Matthias Cosler, Christopher Hahn, Daniel Mendoza, Frederik Schmitt, and Caroline Trippel. 2023. nl2spec: Interactively Translating Unstructured Natural Language to Temporal Logics with Large Language Models. In *CAV*, Vol. 13965. 383–396.

[5] Renzo Degiovanni, Facundo Molina, Germán Regis, and Nazareno Aguirre. 2018. A genetic algorithm for goal-conflict identification. In *ASE*. 520–531.

[6] Yihong Dong, Xue Jiang, Zhi Jin, and Ge Li. 2023. Self-collaboration Code Generation via ChatGPT. *CoRR* abs/2304.07590 (2023).

[7] Alexandre Duret-Lutz, Alexandre Lewkowicz, Amaury Fauchille, Thibaud Michaud, Etienne Renault, and Laurent Xu. 2016. Spot 2.0 - A Framework for LTL and ω -Automata Manipulation. In *ATVA*. 122–129.

[8] Daniel Fried, Armen Aghajanyan, Jessy Lin, Sida Wang, Eric Wallace, Freda Shi, Ruiqi Zhong, Scott Yih, Luke Zettlemoyer, and Mike Lewis. 2023. InCoder: A Generative Model for Code Infilling and Synthesis. In *ICLR*.

[9] Francesco Fuggitti and Tathagata Chakraborti. 2023. NL2LTL - a Python Package for Converting Natural Language (NL) Instructions to Linear Temporal Logic (LTL) Formulas. In *AAAI*. 16428–16430.

[10] Christopher Hahn, Frederik Schmitt, Jens U. Kreber, Markus Norman Rabe, and Bernd Finkbeiner. 2021. Teaching Temporal Logics to Neural Networks. In *ICLR*.

[11] Jianwen Li, Geguang Pu, Lijun Zhang, Moshe Y. Vardi, and Jifeng He. 2018. Accelerating LTL satisfiability checking by SAT solvers. *J. Log. Comput.* 28, 6 (2018), 1011–1030.

[12] Jianwen Li, Lijun Zhang, Shufang Zhu, Geguang Pu, Moshe Y. Vardi, and Jifeng He. 2018. An explicit transition system construction approach to LTL satisfiability checking. *Formal Aspects Comput.* 30, 2 (2018), 193–217.

[13] Jianwen Li, Shufang Zhu, Geguang Pu, Lijun Zhang, and Moshe Y. Vardi. 2019. SAT-based explicit LTL reasoning and its application to satisfiability checking. *Formal Methods in System Design* 54, 2 (2019), 164–190.

[14] Weilin Luo, Hai Wan, Jianfeng Du, Xiaoda Li, Yuze Fu, Rongzhen Ye, and Delong Zhang. 2022. Teaching LTLf Satisfiability Checking to Neural Networks. In *IJCAI*. 3292–3298.

[15] Weilin Luo, Hai Wan, Xiaotong Song, Binhao Yang, Hongzhen Zhong, and Yin Chen. 2021. How to Identify Boundary Conditions with Contrasty Metric?. In *ICSE*. 1473–1484.

[16] Weilin Luo, Hai Wan, Delong Zhang, Jianfeng Du, and Hengdi Su. 2022. Checking LTL Satisfiability via End-to-end Learning. In *ASE*. 21:1–21:13.

[17] Nicolas Markey and Philippe Schnoebelen. 2003. Model Checking a Path. In *CONCUR*, Vol. 2761. 248–262.

[18] OpenAI. 2023. ChatGPT. https://openai.com/blog/chatgpt/

[19] OpenAI. 2023. GPT-4 Technical Report. *CoRR* abs/2303.08774 (2023).

[20] Long Ouyang, Jeffrey Wu, Xu Jiang, Diogo Almeida, Carroll L. Wainwright, Pamela Mishkin, Chong Zhang, Sandhini Agarwal, Katarina Slama, Alex Ray, John Schulman, Jacob Hilton, Fraser Kelton, Luke Miller, Maddie Simens, Amanda Askell, Peter Welinder, Paul F. Christiano, Jan Leike, and Ryan Lowe. 2022. Training language models to follow instructions with human feedback. In *NeurIPS*.

[21] Abulhair Saparov and He He. 2023. Language Models Are Greedy Reasoners: A Systematic Formal Analysis of Chain-of-Thought. In *ICLR*.

[22] Teven Le Scao, Thomas Wang, Daniel Hesslow, Stas Bekman, M. Saiful Bari, Stella Biderman, Hady Elsahar, Niklas Muennighoff, Jason Phang, Ofir Press, Colin Raffel, Victor Sanh, Sheng Shen, Lintang Sutawika, Jaesung Tae, Zheng Xin Yong, Julien Launay, and Iz Beltagy. 2022. What Language Model to Train if You Have One Million GPU Hours?. In *EMNLP*. 765–782.

[23] A. Prasad Sistla and Edmund M. Clarke. 1985. The Complexity of Propositional Linear Temporal Logics. *J. ACM* 32, 3 (1985), 733–749.

[24] Immanuel Trummer. 2022. CodexDB: Synthesizing Code for Query Processing from Natural Language Instructions using GPT-3 Codex. *Proc. VLDB Endow.* 15, 11 (2022), 2921–2928.

[25] Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Brian Ichter, Fei Xia, Ed H. Chi, Quoc V. Le, and Denny Zhou. 2022. Chain-of-Thought Prompting Elicits Reasoning in Large Language Models. In *NeurIPS*.

[26] Thilini Wijesiriwardene, Ruwan Wickramarachchi, Bimal G. Gajera, Shreeyash Mukul Gowaikar, Chandan Gupta, Aman Chadha, Aishwarya Naresh Reganti, Amit P. Sheth, and Amitava Das. 2023. ANALOGICAL - A Novel Benchmark for Long Text Analogy Evaluation in Large Language Models. In *ACL*. 3534–3549.

[27] Shunyu Yao, Dian Yu, Jeffrey Zhao, Izhak Shafran, Thomas L. Griffiths, Yuan Cao, and Karthik Narasimhan. 2023. Tree of Thoughts: Deliberate Problem Solving with Large Language Models. *CoRR* abs/2305.10601 (2023).

[28] Aohan Zeng, Xiao Liu, Zhengxiao Du, Zihan Wang, Hanyu Lai, Ming Ding, Zhuoyi Yang, Yifan Xu, Wendi Zheng, Xiao Xia, Weng Lam Tam, Zixuan Ma, Yufei Xue, Jidong Zhai, Wenguang Chen, Zhiyuan Liu, Peng Zhang, Yuxiao Dong, and Jie Tang. 2023. GLM-130B: An Open Bilingual Pre-trained Model. In *ICLR*.