ASP Chef Chats with Large Language Models

Mario Alviano, **Pietro Macrì**, **Luis Angel Rodriguez Reiners**
DEMACS, University of Calabria, Via Bucci 30/B, 87036 Rende (CS), Italy
{mario.alviano, luis.reiners}@unical.com

Abstract

ASP Chef enriches Answer Set Programming (ASP) with the notion of recipe, that is, a sequence of operations on answer sets. Recipes are designed and executed in modern browsers, and further improve the fast prototyping capabilities of ASP. This paper introduces new operations designed to integrate Large Language Models (LLMs) in recipe, with the aim of combining the reasoning strength of ASP with the natural language capabilities of LLMs, to enable more interactive and adaptive problem-solving workflows. In a nutshell, answer sets in input are transformed into prompts for LLMs, whose responses are processed to extract facts for subsequent operations within the recipe.

1 Introduction

Large Language Models (LLMs) and Answer Set Programming (ASP) are complementary AI paradigms, driving growing interest in their integration [Nye *et al.*, 2021; Yang *et al.*, 2023; Borroto *et al.*, 2024; Ishay *et al.*, 2023a; Ishay *et al.*, 2023b; Rajasekharan *et al.*, 2023a; Rajasekharan *et al.*, 2023b; Kalyanpur *et al.*, 2024; Lin *et al.*, 2024; Brancas *et al.*, 2024; Zeng *et al.*, 2024; Coppolillo *et al.*, 2024]. LLMs have revolutionized natural language processing [Brown and et al., 2020; Chowdhery and et al., 2023; Touvron and et al., 2023], while ASP offers robust capabilities in logical reasoning [Gelfond and Lifschitz, 1990; Marek and Truszczynski, 1999; Niemelä, 1999]. This makes ASP invaluable for applications that demand strong inference capabilities (e.g., [Cardellini *et al.*, 2023; Cardellini *et al.*, 2024b; Cardellini *et al.*, 2024a; Cappanera *et al.*, 2023; Wotawa, 2020; Taupe *et al.*, 2021; Dodaro *et al.*, 2024]) beyond the scope of LLMs [Hadi *et al.*, 2023].

ASP Chef [Alviano *et al.*, 2023; Alviano and Reiners, 2024] improves the user experience of ASP and simplifies its use in various computational tasks. Thanks to its web-based platform, ASP users can create and execute complex workflows, known as *recipes*, which are executed directly within the browser and can include data manipulation and visualization operations. ASP Chef is powered by the WebAssembly version of CLINGO [Gebser *et al.*, 2019] (https://github.com/domoritz/clingo-wasm), and is an ideal tool for education, rapid prototyping, and practical problem solving [Costantini and Formisano, 2024; Böhl *et al.*, 2024; Alviano *et al.*, 2024a; Alviano *et al.*, 2024b; Alviano and Rodriguez Reiners, 2024].

This paper presents an integration of LLMs into ASP Chef that allows users to harness the power of LLMs and ASP within a unified environment. The integration proves particularly useful in the three key areas targeted by ASP Chef. In an educational context, it enables users to quickly generate example data by leveraging the broad knowledge base of LLMs, to facilitate the understanding of ASP concepts and applications. For rapid prototyping, LLMs assist users by providing immediate suggestions on ASP syntax, debugging hints, and clarifications regarding ASP Chef documentation, significantly reducing the time required to construct and refine logic programs. Finally, in practical problem-solving, LLMs can be used to extract structured data from unstructured sources, which is then processed through ASP Chef recipes for logical reasoning; the results of the reasoning process can subsequently be mapped back into natural language (using LLMs) to improve their interpretability.

In a nutshell, we introduce operations to register API keys of LLM servers, to configure endpoints and models to use, and to perform remote *chat completion* requests. Such requests use messages stored in ASP facts and can incorporate mustache queries (introduced here) to dynamically include data from other facts. This approach enables prompts for LLMs to be generated from templates, where placeholders are automatically replaced with the results of mustache queries. As a result, ASP and LLMs can be seamlessly combined, allowing for dynamic and context-aware interactions. The response generated by the LLM is stored as an ASP fact, allowing seamless integration with other ASP Chef operations. For example, if the LLM outputs a response in comma-separated values (CSV) format, the *Parse CSV* operation can transform each value into a structured fact. Subsequently, the *Search Models* operation can process these facts to derive a meaningful relational representation. Alternatively, the usual Markdown format used by LLMs can be processed by the *Markdown* operation of ASP Chef to visualize the generated response as a side output of the recipe.

2 Background

ASP. A program is a set of rules defining conditions (conjunctive bodies) under which atoms must be derived (atomic heads) or guessed (choices). Programs are associated with zero or more answer sets, i.e., interpretations satisfying all rules and a stability condition [Gelfond and Lifschitz, 1990]. Programs are extended with `#show` directives of the form

`#show` $p(\bar{t})$ `:` *conjunctive_query* `.`

where p is an optional predicate, $\bar{t}$ is a possibly empty sequence of terms, and *conjunctive_query* is a conjunction. Answer sets of the program are projected according to the `#show` directives. We refer the ASP-Core-2 format for details [Calimeri *et al.*, 2020].

Example 1. *The following program solves the graph coloring problem in ASP:*

```
node(A) :- edge(A,B). node(B) :- edge(A,B).
{assign(N,C) : color(C)} = 1 :- node(N).
:- edge(X,Y), assign(X,C), assign(Y,C).
#show (N,C) : assign(N,C).
```

The first line contains two rules defining nodes from edges. The second line is a choice rule guessing one color for each node. The third line is a constraint enforcing different colors for adjacent nodes. Finally, the fourth line is a `#show` directive projecting the answer sets over the node–color assignment. Given `color(red)` and `color(green)`, the graph `edge(a,b)`, `edge(a,c)`, `edge(b,d)`, `edge(c,d)` has 2 (projected) answer sets, among them `(a,red)`, `(b,green)`, `(c,green)`, `(d,red)`. ■

ASP Chef. An *operation* O is a function receiving in input a sequence of interpretations and producing in output a sequence of interpretations. Operations may produce side outputs (e.g., a graph visualization) and accept parameters to influence their behavior. An *ingredient* is an instantiation of a parameterized operation with side output. A *recipe* is a tuple of the form $(encode, Ingredients, decode)$, where $Ingredients$ is a (finite) sequence $O_1\langle P_1\rangle, \ldots, O_n\langle P_n\rangle$ of ingredients, and $encode$ and $decode$ are Boolean values. If $encode$ is true, the input of the recipe is mapped to [[`__base64__("`s`")`]], where $s = Base64(s_{in})$ (i.e., the Base64–encoding of the input string s_{in}). After that, the ingredients are applied one after another. Finally, if $decode$ is true, every occurrence of `__base64__(`s`)` is replaced with (the ASCII string associated with) $Base64^{-1}(s)$.

Large Language Models (LLMs). LLMs are AI systems designed to process and generate human-like text. Here, LLMs are used as black box operators on text (functions that take text in input and produce text in output). The text in input is called *prompt*, and the text in output is called *generated text* or *response*. The prompt is a sequence of messages from three roles, namely *system*, *user* and *assistant*. System messages set behavior, tone, and context for the assistant. User messages represent queries to or instructions for the assistant. Assistant messages are responses to user queries.

3 LLMs Operations

Config. The *@LLMs/Config* operation extends each input interpretation with facts representing parameters like server, model and temperature. These facts have the form `__llms_config__(key,"value")`, where `__llms_config__` can be set in the ingredient. The temperature is expressed as a percentage (to accommodate differences between servers), with 0% disabling randomness.

Chat Completion. For each input interpretation I, the *@LLMs/Chat Completion* takes the configuration from instances of `__llms_config__` (or the predicate specified in the ingredient), and messages from atoms of the form `__message__(role("content"))`, where `role` is `system`, `user` or `assistant`, and `content` is a string with *mustache queries*, defined next. A mustache query has the form `{{ `Π` }}`, where Π is an ASP program with `#show` directives, and it is replaced by one projected answer set of $\Pi \cup \{p(\bar{t}). \mid p(\bar{t}) \in I\}$. In more details, only tuples of terms are left, and specific atoms are interpreted as follows: `separator/1` defines the separator for tuples (default `\n`); `term_separator/1` defines the separator between terms (default `,`); `prefix/1` and `suffix/1` define strings to be added before and after each tuple (empty by default); `ol/1` and `ul/1` to produce ordered and unordered lists, and `th` and `tr` to produce tables (in Markdown); `base64` to decode Base64-encoded strings. Moreover,

`{{= (terms): conjunctive_query }}`

is syntactic sugar for

`{{ #show (terms): conjunctive_query. }}`.

The response given by the LLM is Base64-encoded in the predicate `__base64__` (or the predicate specified in the ingredient), and can be further processed by the subsequent ingredients in the recipe.

Example 2. *Let I be the interpretation comprising `assign(a,red)`, `assign(b,green)`, `assign(c,green)`, `assign(d,red)`. Let I also contain the following instances of `__message__/1`: `system("If you are unsure, say \"I don't know\"")` and `user("{{ `Π` }}\n What's the color of node a?")`, where Π is*

```
#show (N,C) : assign(N,C).
#show prefix("Node ").
#show term_separator(" has color ").
#show suffix(".").
```

After mustache query replacement, the user message is

```
Node a has color red.
Node b has color green.
Node c has color green.
Node d has color red.

What's the color of node a?
```

If I also contains configuration atoms for using Groq [Abts et al.*, 2022] and the model llama3-70b-8192 with temperature 0%, the resulting response `"The color of node a is red."` is Base64-encoded in the output predicate.* ■

To ease the representation of prompts, especially those including mustache queries, the *@LLMs/Chat Completion* operation accepts the additional parameters *system role*, *user role* and *assistant role*, which specifies unary predicates whose terms are Base64-encoded.

Register API Key. Servers usually expect an API key to be provided with each request. In ASP Chef, such API keys are retrieved from the session storage, where they are saved via a *@LLMs/Register API Key* ingredient. This way, API keys are not part of the recipe, are not accessible by other tabs of the browser, and are forgotten when the tab with the recipe is closed. Alternatively, API keys can be stored permanently in the local storage, again with a *@LLMs/Register API Key* ingredient. In this case, a recipe can copy an API key from the local storage to the session storage, hence enabling it in the current session, if the user confirm this intention.

Unregister API Keys. API keys enabled in the current session can be disabled via a *@LLMs/Unregister API Keys*, which also lists the permanently stored API keys and gives the possibility to remove them selectively or in block.

4 Use Cases

We report a few use cases whose recipes are available online at https://asp-chef.alviano.net/s/LLMs.

4.1 Generate Example Data

ASP is often introduced by showing how to address combinatorial puzzles such as Sudoku. Manually writing input facts can be tedious, while generating them with a script may introduce an additional barrier for learners. As an alternative, we can ask an LLM for a random instance, using the system message **"Your answer must be in CSV format. Just CSV and nothing else! Use TAB and new lines as separators."** and the user message

```
Give me a random instance of {{= S :
  size(S) }}x{{= S : size(S) }} Sudoku.
Each known cell must have a number between
  1 and {{= S : size(S) }}.
Use 0 for the unknown cells.
```

where `size(9)` can be a given fact. The generated instance can then be processed by *Parse CSV* and solved with ASP.

For another example, we can ask for factual data about cities (and do some reasoning on such data using ASP):

```
__message__(system("Answer with CSV only.
  Use TAB as separator.")).
__message__(user("Give me a list of cities
  and their population.")).
```

4.2 Interacting with ASP Chef Documentation

The *Documentation* operation can be used to consult the ASP Chef documentation, but also to generate facts storing such documentation. This approach enables the possibility to include documentation snippets in messages for LLMs, for example using the user message

```
Give me an example of the usage of Merge
  and Split. Their documentation follow.
{{ #show base64(X) : __doc__(X).
   #show separator("\n\n").      }}
```

The documentation of *Merge* and *Split*, stored in instances of `__doc__/1`, reaches the LLM, which is then capable of generating an example even if it has no previous knowledge on the two operations. The generated answer can be shown with the *Markdown* operation.

4.3 Extract Structured Data

We are designing a package delivery route for a small courier company. A delivery driver starts at a given point and needs to reach another point, making sure to visit every location exactly once. Connections are also given. For example,

```
Starts at Point A and  reach Point G.
- A is connected to B and C
- B is connected to A, C and D
- C is connected to A and D
- D is connected to B, C, and E
- E is connected to D, F and H
- F is connected to E and G
- G is connected to F and H
- H is connected to F
```

The above input can be stored in a `__base64__` predicate and combined with the following user message:

```
{{= base64(X) : __base64__(X) }}
----
Give me the start point in the first line,
and the reach point in the second line.
After that, list all connections,
one per line.
```

Once input data is structured in CSV, *Parse CSV* and *Search Models* can easily obtain a relational representation in predicates `start/1`, `target/1`, `node/1` and `link/2`. After that, a Hamiltonian path (if any) is obtained with the following ASP program:

```
reach(X) :- start(X).
reach(Y) :- next(X,Y).
{next(X,Y) : link(X,Y)} = 1 :-
    reach(X), not target(X).
:- next(X,Y), next(Z,Y), X < Z.
:- node(X), not reach(X).
```

4.4 Improve Interpretability of Reasoning Results

SELinux policies define access control rules that govern interactions between processes, files, and other system resources. These policies are typically very large rule sets, making it challenging to manually analyze permissions for a specific subject type and security context. We can efficiently filter SELinux policies using ASP, and provide human-readable insights into what a given user or process can do in a specific security context using LLMs. Specifically, we can use the system message **"You are an expert in SELinux policy management and your task is to create a detailed text starting from a specific set of policies."** and pack the user message using

```
__message__(user(@string_format(
  "allow %s %s:%s %s;", S,O,C,P))
) :- needed_policy(S,O,C,P).
```

where `needed_policy` contains the filtered input facts.

5 Conclusion

The integration of LLMs into ASP Chef bridges the gap between human-intuitive input and machine-processable ASP facts, opening new possibilities for reasoning, knowledge representation, and automated decision-making.

Ethical Statement

The use of LLMs inherently exposes to risks related to transparency and accountability. Users of @LLMs operations of ASP Chef must be clearly informed about such risks.

Acknowledgments

This work was supported by the Italian Ministry of University and Research (MUR) under PRIN project PRODE "Probabilistic declarative process mining", CUP H53D23003420006, under PNRR project FAIR "Future AI Research", CUP H23C22000860006, under PNRR project Tech4You "Technologies for climate change adaptation and quality of life improvement", CUP H23C22000370006, and under PNRR project SERICS "SEcurity and RIghts in the CyberSpace", CUP H73C22000880001; by the Italian Ministry of Health (MSAL) under POS projects CAL.HUB.RIA (CUP H53C22000800006) and RADIOAMICA (CUP H53C22000650006); by the Italian Ministry of Enterprises and Made in Italy under project STROKE 5.0 (CUP B29J23000430005); under PN RIC project ASVIN "Assistente Virtuale Intelligente di Negozio" (CUP B29J24000200005); and by the LAIA lab (part of the SILA labs). Mario Alviano is member of Gruppo Nazionale Calcolo Scientifico-Istituto Nazionale di Alta Matematica (GNCS-INdAM).

References

[Abts *et al.*, 2022] Dennis Abts, John Kim, Garrin Kimmell, Matthew Boyd, Kris Kang, Sahil Parmar, Andrew C. Ling, Andrew Bitar, Ibrahim Ahmed, and Jonathan Ross. The groq software-defined scale-out tensor streaming multiprocessor : From chips-to-systems architectural overview. In *2022 IEEE Hot Chips 34 Symposium, HCS 2022, Cupertino, CA, USA, August 21-23, 2022*, pages 1–69. IEEE, 2022.

[Alviano and Reiners, 2024] Mario Alviano and Luis Angel Rodriguez Reiners. ASP chef: Draw and expand. In Pierre Marquis, Magdalena Ortiz, and Maurice Pagnucco, editors, *Proceedings of the 21st International Conference on Principles of Knowledge Representation and Reasoning, KR 2024, Hanoi, Vietnam. November 2-8, 2024*, 2024.

[Alviano and Rodriguez Reiners, 2024] Mario Alviano and Luis Angel Rodriguez Reiners. Integrating MiniZinc with ASP Chef: Browser-based constraint programming for education and prototyping. In *International Conference on Logic Programming and Nonmonotonic Reasoning*, pages 174–186. Springer, 2024.

[Alviano *et al.*, 2023] Mario Alviano, Davide Cirimele, and Luis Angel Rodriguez Reiners. Introducing ASP recipes and ASP chef. In *ICLP Workshops*, volume 3437 of *CEUR Workshop Proceedings*. CEUR-WS.org, 2023.

[Alviano *et al.*, 2024a] Mario Alviano, Danilo Amendola, and Luis Angel Rodriguez Reiners. Addressing marketplace logistic tasks in answer set programming. *Intelligenza Artificiale*, 18(2):261–278, 2024.

[Alviano *et al.*, 2024b] Mario Alviano, Paola Guarasci, Luis Angel Rodriguez Reiners, and Ilaria R Vasile. Integrating structured declarative language (sdl) into ASP Chef. In *International Conference on Logic Programming and Nonmonotonic Reasoning*, pages 387–392. Springer, 2024.

[Böhl *et al.*, 2024] Elisa Böhl, Stefan Ellmauthaler, and Sarah Alice Gaggl. Winning snake: Design choices in multi-shot asp. *Theory and Practice of Logic Programming*, 24(4):772–789, 2024.

[Borroto *et al.*, 2024] Manuel Borroto, Irfan Kareem, and Francesco Ricca. Towards automatic composition of asp programs from natural language specifications. *arXiv preprint arXiv:2403.04541*, 2024.

[Brancas *et al.*, 2024] Ricardo Brancas, Vasco Manquinho, and Ruben Martins. Combining logic with large language models for automatic debugging and repair of asp programs, 2024.

[Brown and et al., 2020] Tom B. Brown and et al. Language models are few-shot learners. *CoRR*, abs/2005.14165, 2020.

[Calimeri *et al.*, 2020] Francesco Calimeri, Wolfgang Faber, Martin Gebser, Giovambattista Ianni, Roland Kaminski, Thomas Krennwallner, Nicola Leone, Marco Maratea, Francesco Ricca, and Torsten Schaub. Asp-core-2 input language format. *Theory Pract. Log. Program.*, 20(2):294–309, 2020.

[Cappanera *et al.*, 2023] Paola Cappanera, Marco Gavanelli, Maddalena Nonato, and Marco Roma. Logic-based benders decomposition in answer set programming for chronic outpatients scheduling. *Theory Pract. Log. Program.*, 23(4):848–864, 2023.

[Cardellini *et al.*, 2023] Matteo Cardellini, Carmine Dodaro, Giuseppe Galatà, Anna Giardini, Marco Maratea, Nicholas Nisopoli, and Ivan Porro. Rescheduling rehabilitation sessions with answer set programming. *J. Log. Comput.*, 33(4):837–863, 2023.

[Cardellini *et al.*, 2024a] Matteo Cardellini, Carmine Dodaro, Marco Maratea, and Mauro Vallati. Optimising dynamic traffic distribution for urban networks with answer set programming. *Theory Pract. Log. Program.*, 24(4):825–843, 2024.

[Cardellini *et al.*, 2024b] Matteo Cardellini, Paolo De Nardi, Carmine Dodaro, Giuseppe Galatà, Anna Giardini, Marco Maratea, and Ivan Porro. Solving rehabilitation scheduling problems via a two-phase ASP approach. *Theory Pract. Log. Program.*, 24(2):344–367, 2024.

[Chowdhery and et al., 2023] Aakanksha Chowdhery and et al. Palm: Scaling language modeling with pathways. *J. Mach. Learn. Res.*, 24:240:1–240:113, 2023.

[Coppolillo *et al.*, 2024] Erica Coppolillo, Francesco Calimeri, Giuseppe Manco, Simona Perri, and Francesco Ricca. LLASP: fine-tuning large language models for answer set programming. In Pierre Marquis, Magdalena Ortiz, and Maurice Pagnucco, editors, *Proceedings of the*

21st International Conference on Principles of Knowledge Representation and Reasoning, KR 2024, Hanoi, Vietnam. November 2-8, 2024, 2024.

[Costantini and Formisano, 2024] Stefania Costantini and Andrea Formisano. Solver fast prototyping for reduct-based ELP semantics. In Emanuele De Angelis and Maurizio Proietti, editors, *Proceedings of the 39th Italian Conference on Computational Logic, Rome, Italy, June 26-28, 2024*, volume 3733 of *CEUR Workshop Proceedings*. CEUR-WS.org, 2024.

[Dodaro *et al.*, 2024] Carmine Dodaro, Giuseppe Galatà, Martin Gebser, Marco Maratea, Cinzia Marte, Marco Mochi, and Marco Scanu. Operating room scheduling via answer set programming: Improved encoding and test on real data. *J. Log. Comput.*, 34(8):1556–1579, 2024.

[Gebser *et al.*, 2019] Martin Gebser, Roland Kaminski, Benjamin Kaufmann, and Torsten Schaub. Multi-shot ASP solving with clingo. *Theory Pract. Log. Program.*, 19(1):27–82, 2019.

[Gelfond and Lifschitz, 1990] Michael Gelfond and Vladimir Lifschitz. Logic programs with classical negation. In D. Warren and Peter Szeredi, editors, *Logic Programming: Proc. of the Seventh International Conference*, pages 579–597, 1990.

[Hadi *et al.*, 2023] Muhammad Usman Hadi, Rizwan Qureshi, Abbas Shah, Muhammad Irfan, Anas Zafar, Muhammad Bilal Shaikh, Naveed Akhtar, Jia Wu, Seyedali Mirjalili, et al. Large language models: a comprehensive survey of its applications, challenges, limitations, and future prospects. *Authorea Preprints*, 2023.

[Ishay *et al.*, 2023a] Adam Ishay, Zhun Yang, and Joohyung Lee. Leveraging large language models to generate answer set programs. In *Proceedings of the 20th International Conference on Principles of Knowledge Representation and Reasoning*, pages 374–383, 2023.

[Ishay *et al.*, 2023b] Adam Ishay, Zhun Yang, and Joohyung Lee. Leveraging Large Language Models to Generate Answer Set Programs. In *Proceedings of the 20th International Conference on Principles of Knowledge Representation and Reasoning*, pages 374–383, 8 2023.

[Kalyanpur *et al.*, 2024] Aditya Kalyanpur, Kailash Saravanakumar, Victor Barres, Jennifer Chu-Carroll, David Melville, and David A. Ferrucci. LLM-ARC: enhancing llms with an automated reasoning critic. *CoRR*, abs/2406.17663, 2024.

[Lin *et al.*, 2024] Xinrui Lin, Yangfan Wu, Huanyu Yang, Yu Zhang, Yanyong Zhang, and Jianmin Ji. CLMASP: coupling large language models with answer set programming for robotic task planning. *CoRR*, abs/2406.03367, 2024.

[Marek and Truszczynski, 1999] Victor W. Marek and Miroslaw Truszczynski. Stable models and an alternative logic programming paradigm. In *The Logic Programming Paradigm*, Artificial Intelligence, pages 375–398. Springer, 1999.

[Niemelä, 1999] Ilkka Niemelä. Logic programs with stable model semantics as a constraint programming paradigm. *Ann. Math. Artif. Intell.*, 25(3-4):241–273, 1999.

[Nye *et al.*, 2021] Maxwell Nye, Michael Tessler, Josh Tenenbaum, and Brenden M Lake. Improving coherence and consistency in neural sequence models with dual-system, neuro-symbolic reasoning. *Advances in Neural Information Processing Systems*, 34:25192–25204, 2021.

[Rajasekharan *et al.*, 2023a] Abhiramon Rajasekharan, Yankai Zeng, and Gopal Gupta. Argument analysis using answer set programming and semantics-guided large language models. In *ICLP Workshops*, 2023.

[Rajasekharan *et al.*, 2023b] Abhiramon Rajasekharan, Yankai Zeng, Parth Padalkar, and Gopal Gupta. Reliable natural language understanding with large language models and answer set programming. In Enrico Pontelli, Stefania Costantini, Carmine Dodaro, Sarah Alice Gaggl, Roberta Calegari, Artur S. d'Avila Garcez, Francesco Fabiano, Alessandra Mileo, Alessandra Russo, and Francesca Toni, editors, *Proceedings 39th International Conference on Logic Programming, ICLP 2023, Imperial College London, UK, 9th July 2023 - 15th July 2023*, volume 385 of *EPTCS*, pages 274–287, 2023.

[Taupe *et al.*, 2021] Richard Taupe, Gerhard Friedrich, Konstantin Schekotihin, and Antonius Weinzierl. Solving configuration problems with ASP and declarative domain specific heuristics. In Michel Aldanondo, Andreas A. Falkner, Alexander Felfernig, and Martin Stettinger, editors, *Proceedings of the 23rd International Configuration Workshop (CWS/ConfWS 2021), Vienna, Austria, 16-17 September, 2021*, volume 2945 of *CEUR Workshop Proceedings*, pages 13–20. CEUR-WS.org, 2021.

[Touvron and et al., 2023] Hugo Touvron and et al. Llama: Open and efficient foundation language models. *CoRR*, abs/2302.13971, 2023.

[Wotawa, 2020] Franz Wotawa. On the use of answer set programming for model-based diagnosis. In Hamido Fujita, Philippe Fournier-Viger, Moonis Ali, and Jun Sasaki, editors, *Trends in Artificial Intelligence Theory and Applications. Artificial Intelligence Practices - 33rd International Conference on Industrial, Engineering and Other Applications of Applied Intelligent Systems, IEA/AIE 2020, Kitakyushu, Japan, September 22-25, 2020, Proceedings*, volume 12144 of *Lecture Notes in Computer Science*, pages 518–529. Springer, 2020.

[Yang *et al.*, 2023] Zhun Yang, Adam Ishay, and Joohyung Lee. Coupling large language models with logic programming for robust and general reasoning from text. In *Findings of the Association for Computational Linguistics: ACL 2023*, pages 5186–5219, 2023.

[Zeng *et al.*, 2024] Yankai Zeng, Abhiramon Rajasekharan, Kinjal Basu, Huaduo Wang, Joaquín Arias, and Gopal Gupta. A reliable common-sense reasoning socialbot built using llms and goal-directed asp. *Theory and Practice of Logic Programming*, 24(4):606–627, 2024.