Revisiting Structured Representation Use in LLM

Michal Chudoba[1][0009−0001−2233−4074], Tomasz Wiktorski[1][0000−0002−5940−8102], and Sergey Alyaev[2][0000−0002−2105−2067]

[1] University of Stavanger, 4036 Stavanger, Norway
{michal.chudoba,tomasz.wiktorski}@uis.no
[2] NORCE Norwegian Research Centre, 5838 Bergen, Norway
saly@norceresearch.no

Abstract. Structured representations (SRs), pivotal in pre-LLM NLP, play a contentious role today, with studies showing that they can degrade task performance. The prevailing hypothesis suggests that Large Language Models (LLMs) are unfamiliar with traditional SR formats like Abstract Meaning Representation (AMR). We argue that this view is incomplete, as LLMs are extensively trained on structured data, particularly programming code. To test this, we introduce two new prompt frameworks and evaluate three representation formats (AMR vs. RDF vs. Python code) across multiple LLMs and tasks. Our findings indicate that the choice of representation is of high importance. Across multiple models and tasks, we show that Python code and RDF outperform AMR up to 20% in classification tasks. The effectiveness of any SR is also conditioned on the LLM's baseline capability, the prompting method, and the quality of the representation itself. Although SRs can substantially boost performance for models with weaker baselines, they offer diminishing returns and can harm performance for models that are already highly capable, confirming a 'sweet spot' for their application. Our work demonstrates that the utility of SRs in the LLM era depends on their alignment with the models' training data.

Keywords: Natural Language Processing · Large Language Models · Structured Representation

1 Introduction

Recent work has questioned the role of structured representations (SR) in LLM-based NLP [11, 30], studies showing performance degradation when incorporating traditional SR formats such as Abstract Meaning Representation (AMR) [2], First Order Logic (FOL) [3] and Parse Syntax Trees (PST) [10]. The prevailing hypothesis attributes this to LLMs' unfamiliarity with such representations.

We argue that this interpretation is misplaced. Rather than SRs being inherently ineffective, we argue that the issue lies specifically with AMR, FOL, and PST which are rarely seen in LLM training data. LLMs are extensively trained on structured information, particularly programming code present in GitHub repositories which represent part of current LLMs pre-training data (4.5% of

LLaMA, 5% of PaLM [25, 5]). We test this by comparing AMR with RDF [16] and Python code representations across multiple tasks and models, re-examining AMRCoT [11] and SR-LLM [30] frameworks while introducing two new variants.

Our findings show that the enhanced effectiveness of Python code and RDF indeed points to the importance of aligning SRs with LLM training data. However, the utility of all SRs is conditioned on the model's baseline capability, prompting method, and representation quality.

The contribution of this work is threefold.

- We provide a comprehensive analysis of the effectiveness of structured representations in LLMs, highlighting the factors that influence their performance.
- We introduce new prompting frameworks that leverage insights from recent research on irrelevant context, improving the integration of structured representations in LLMs.
- We demonstrate the potential of using programming code as a structured representation, opening new avenues for research in this area.

The code and prompts to reproduce our experiments are available in the Github repository jinymusim/sr_in_llm[3].

2 Related Literature

2.1 Structured Representations in NLP

Before the rise of LLMs, structured representations such as AMR [2], FOL [3] and PST [10] played a crucial role in various NLP tasks, including summarization [15], knowledge base question answering [12], and detection of toxic content [7].

Recent evaluations in the LLM era question their continued effectiveness. Jin et al. [11] report that incorporating AMR can degrade performance, possibly due to unfamiliarity with its format. Raut et al. [20] find that AMR loses effectiveness beyond three examples in a few-shot prompts (SAMSum [9]). However, Bohao Yang et al. [28] found success with AMR for Dialog Evaluation. Wein and Schneider [27] show that AMR can reduce translation artifacts, though sometimes at the expense of fluency. Zhang et al. [30] observe that SRs can help weaker LLMs but degrade performance in stronger ones, suggesting that highly capable models may already internalize structural reasoning, making explicit SRs redundant or distracting.

2.2 Code as Structured Representation

As noted above, programming code appears in substantial amounts in the LLM training corpora. Being inherently structured, code may be more effective than traditional SR formats such as AMR. Aryabumi et al. [1] show code pre-training

[3] https://github.com/jinymusim/sr_in_llm

improves not just programming but general capabilities including reasoning. Yang et al. [29] report similar findings for Chain-of-Thought prompting. Code4Struct [26] achieves state-of-the-art event structure prediction, while Li et al. [14] demonstrate code with non-English comments improves multilingual reasoning.

3 Methodology

We evaluate a range of prompting-based methods that incorporate SRs into LLM reasoning. In addition to reimplementing approaches such as AMRCoT [11] and SR-LLM [30], we introduce two new variants, SRCot and SR-Triples, designed to test the role of structure presentation and representation format. Across all methods, we experiment with three types of SR: AMR, RDF, and Python Code representations.

3.1 Baseline

As a baseline, we used zero-shot prompting without SRs. The prompts for PubMed45, PAWS and WMT16 are from AMRCoT [11]. Prompt for AgNews is constructed in a similar way.

3.2 AMRCoT

The AMRsCoT method is implemented as in [11], injecting AMRs into the prompts alongside task input in zero-shot setting. Prompts reuse the original work for PubMed45, PAWS, and WMT16. Prompt for AgNews is constructed analogously.

3.3 SR-LLM

We reimplement SR-LLM [30], where structured inputs are instantiated, transformed to natural language descriptions, and integrated into prompts. Since the authors only provide SNLI examples, we construct zero-shot prompts for our datasets following their template.

3.4 SRCot

SRCot builds on AMRCoT but incorporates explicit instructions to ignore irrelevant context [22] and wraps SRs in code blocks for clearer separation.

3.5 SR-Triples

SR-Triples modifies SR-LLM by removing natural language conversion, inserting instantiated triples directly into the prompts to compare effectiveness of transformations versus semi-structured formats.

3.6 Structured Representation Types

We experiment with three types of structured representations:

- **AMR:** A graph-based semantic formalism in PENMAN notation [13].
- **RDF:** A standard triple-based format for relational knowledge without namespaces.
- **Python Code:** Code-based encoding mapping text to Python objects with entity/relation fields.

4 Datasets

We evaluated four datasets covering classification and translation tasks, following AMRCoT [11] and SR-LLM [30]:

- **WMT16** [4]: Machine translation (English→German). Requires preserving semantic meaning across languages while maintaining fluency.
- **PAWS** [32]: Paraphrase identification (binary: paraphrase/non-paraphrase). Requires distinguishing subtle semantic differences despite surface similarity.
- **PubMed45** [8]: Biomedical interaction classification (3 classes: Invalid interaction, Valid interaction, Valid interaction if swapped). Requires understanding complex biomedical relationships.
- **AgNews** [31]: News categorization (4 classes: World, Sports, Business, Sci/Tech). Requires topic identification from the news text.

We sample 500 instances per dataset for computational efficiency while maintaining statistical validity.

4.1 SR Generation

To generate structured representations, we adopt a three-shot prompting strategy inspired by SR-LLM [30]. Generated AMRs are validated using PENMAN parsers, RDF triples are syntactically checked, and Python Code representations are verified using Python's `AST` module. All SRs are generated with GPT-4o-mini [17].

5 Experiment Results

We evaluated five models of varying sizes and capabilities: GPT-4o-mini [17], GPT-4.1-nano [18], Granite-3.3 2B [23], Qwen2.5 3B [24], and gpt-oss-20B [19].

Table 1 shows that the incorporation of AMRCoT generally degrades performance relative to baselines, particularly in translation (WMT16) and biomedical classification (PubMed45). The results show improvements for weaker models (GPT-4.1-nano on PubMed45, Qwen2.5 on PAWS and AgNews), while the performance of stronger models degrades.

Model	Method	WMT16 (BLEU)	PAWS (F1)	PubMed45 (F1)	AgNews (F1)
GPT-4o-mini	Baseline	**35.4**	**80.6**	**62.2**	**84.8**
	AMRCoT	35.1	75.6	48.2	82.4
GPT-4.1-nano	Baseline	**32.7**	**72.4**	50.6	76.4
	AMRCoT	20.5	67.4	**59.8**	**83.2**
Granite-3.3 2B	Baseline	**22.0**	**55.4**	**38.6**	**75.6**
	AMRCoT	18.8	54.6	19.2	75.2
Qwen2.5 3Bb	Baseline	**18.5**	69.0	**71.8**	71.6
	AMRCoT	16.6	**73.6**	66.4	**75.2**
gpt-oss-20B	Baseline	**29.5**	**83.0**	72.4	**83.4**
	AMRCoT	29.0	80.2	**73.6**	82.4

Table 1. Baseline vs. AMRCoT results

Model	SR	WMT16 (BLEU)	PAWS (F1)	PubMed45 (F1)	AgNews (F1)
GPT-4o-mini	AMR	**35.4**	77.4	63.8	84.0
	RDF	35.3	79.6	**74.4**	84.0
	Code	**35.4**	**81.4**	73.2	**84.4**
GPT-4.1-nano	AMR	32.6	66.4	68.0	71.6
	RDF	**32.8**	**73.4**	**76.2**	**81.8**
	Code	32.5	72.0	67.4	77.4
Granite-3.3 2B	AMR	**20.3**	61.2	59.6	65.4
	RDF	20.2	**63.0**	64.4	**66.8**
	Code	20.2	56.6	**73.8**	65.4
Qwen2.5 3B	AMR	**19.2**	**72.6**	64.0	73.8
	RDF	19.1	71.0	**76.8**	**74.4**
	Code	18.8	71.6	74.6	73.8
gpt-oss-20B	AMR	29.2	**84.2**	**71.8**	82.0
	RDF	29.1	83.2	71.2	**82.6**
	Code	**29.8**	84.0	69.8	81.4

Table 2. SRCoT results across AMR, RDF, and Code.

Model	SR	WMT16 (BLEU)	PAWS (F1)	PubMed45 (F1)	AgNews (F1)
GPT-4o-mini	AMR	30.6	**79.0**	62.4	83.6
	RDF	**32.5**	78.2	**67.6**	**84.2**
GPT-4.1-nano	AMR	29.3	72.0	72.2	80.6
	RDF	**29.7**	**73.2**	**75.2**	80.6
Granite-3.3 2B	AMR	13.1	61.2	33.4	69.2
	RDF	**13.9**	**64.0**	**45.6**	**71.0**
Qwen2.5 3B	AMR	17.3	74.8	57.2	72.4
	RDF	**17.4**	**75.8**	**71.0**	**73.0**
gpt-oss-20B	AMR	28.1	**82.4**	71.6	81.0
	RDF	**28.6**	81.6	**72.0**	**81.8**

Table 3. SR-LLM results: comparison between AMR and RDF.

Table 2 reveals that when AMR outperforms RDF or Code, improvements are marginal. In contrast, RDF and Code often deliver substantial gains: +10.6 points on PubMed45 for GPT-4o-mini, +14.2 for Granite-3.3, and +12.8 for Qwen2.5, suggesting that Code and RDF offer more robust benefits than AMR.

Table 3 reinforces the pattern: RDF consistently outperforms AMR. Notable improvements include PubMed45 with Granite-3.3 (+12.2), Qwen2.5 (+13.8) and GPT-4o-mini (+5.2). Importantly, losses of RDF to AMR are very marginal, with the largest being less than 1 point. This shows that we can simply replace AMR with RDF.

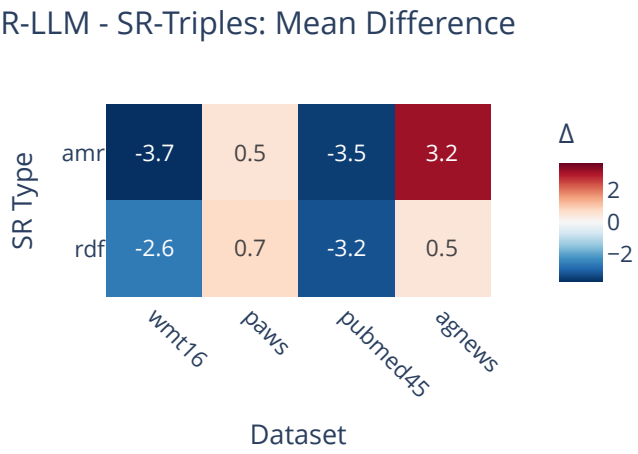

Fig. 1. SR-LLM vs SR-Triples Results capturing the effects of transforming instantiated triples to natural language sentences using templates and LLM refinement.

Figure 1 shows that transforming instantiated triples into natural language (SR-LLM) can sometimes be counterproductive. The tests confirm that SR-Triples outperforms SR-LLM on WMT16 and PubMed45, while other tasks show comparable performance, suggesting that this natural language conversion is mostly unnecessary and sometimes detrimental processing step and that just the translation to a more common format, such as RDF, is preferable.

Figure 2 compares SR types within SRCoT. Both Code and RDF consistently outperform AMR across classification tasks. This aligns with our hypothesis that LLMs are more familiar with representations frequently appearing in pretraining corpora (code, RDF) than with AMR.

Finally, Figure 3 compares GPT-4.1-nano when generating its own SRs versus using those generated by GPT-4o-mini. Although AMR results remain inconclusive, GPT-4o-mini's RDF and Code outputs appear more effective, suggesting that higher-quality SR generation can have a measurable downstream effect.

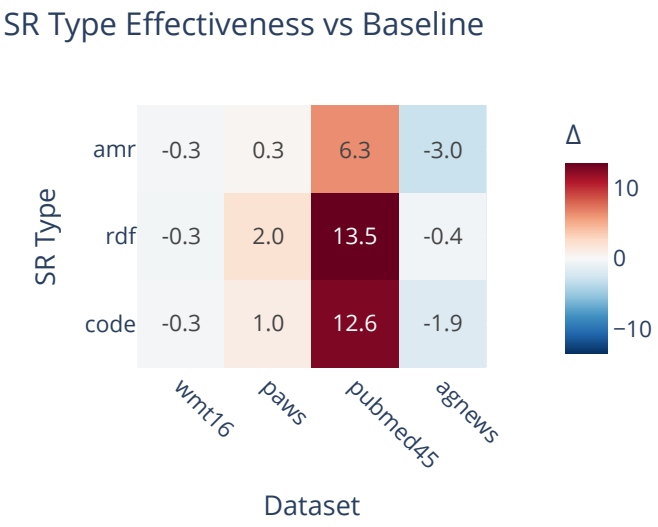

Fig. 2. Comparison of SR types (AMR, RDF, Code) within SRCoT method across datasets and models. Both Code and RDF consistently outperform AMR, confirming SR choice strongly affects performance.

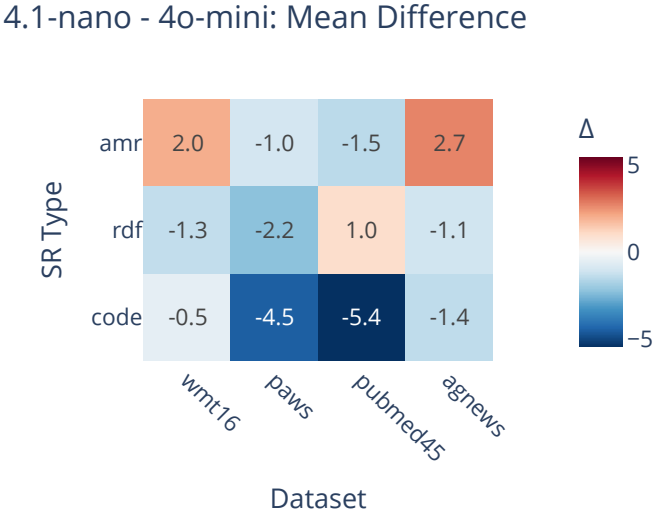

Fig. 3. Comparison of GPT-4.1-nano results with its own SR generation versus the original by GPT-4o-mini

6 Discussion

Our results challenge the broad claim that SRs are ineffective for LLMs. Instead, we find that the issue is specific to certain SR formats. The utility of SRs depends on four factors: (1) alignment with training data, (2) baseline capacity of the model, (3) prompting method, and (4) SR quality.

Alignment with training data matters most. Python code and RDF consistently outperform AMR across tasks and models. This aligns with the composition of the training data: GitHUb repositories with code are common (5% of the major LLM corpora), while AMR/FOL/parse trees are rare. Our findings contradict the hypothesis that SRs in general are ineffective [11, 30]; rather, *specific* SRs (AMR, FOL, PST) are problematic due to unfamiliarity.

The 'sweet spot' for SRs. For models with already strong baselines (GPT-4o-mini: 84.8 F1 on AgNews), injecting an explicit SR appears to add noise rather than signal, degrading performance. For weak baselines (Granite-3.3: 38.6 F1 on PubMed45), SRs provide substantial improvements (Especially SRCoT). This confirms Zhang et al.'s [30] observation but refines it: familiar SRs (code, RDF) help across more model capabilities than unfamiliar ones (AMR).

Task-specific benefits. Knowledge-intensive tasks (PubMed45 biomedical relations, PAWS paraphrase detection) benefit the most from SRs, while fluency-focused tasks (WMT16 translation) show mixed results. This suggests that SRs are most effective when explicit relational structure addresses knowledge gaps in the model rather than the stylistic requirements.

Comparison to the state-of-the-art. While there are specialized systems for individual tasks [6, 21], our focus is on prompt-based zero-shot methods. Within this constraint, our RDF and Code approaches match or exceed the performance of SR-LLM [30] without requiring natural language conversion, offering simpler and often more effective alternatives.

Limitations

First, while we focus on specific SR formats (AMR, RDF, Code) and tasks, the SR landscape is vast. The tasks were selected based on previous literature [11, 30] to allow direct comparison.

Second, our experiments involve only English text. The effectiveness of SRs can differ across languages, particularly given that the advantages of the code can vary with language-specific training data distributions.

Third, following previous work, the generation of SR was performed using a general model (GPT-4o-mini) rather than specialized parsers. While we validate syntax (PENMAN for AMR, AST for code), semantic correctness is not guaranteed. Figure 3 suggests that SR quality affects downstream performance, indicating that specialized generators could yield different results.

References

1. Aryabumi, V., Su, Y., et al.: To code, or not to code? exploring impact of code in pre-training. arXiv preprint arXiv:2408.10914 (2024)
2. Banarescu, L., Bonial, C., et al.: Abstract Meaning Representation for sembanking. In: Proc. of LAW. pp. 178–186 (2013)
3. Barwise, J.: An introduction to first-order logic. In: Studies in Logic and the Foundations of Mathematics, vol. 90, pp. 5–46 (1977)
4. Bojar, O., Chatterjee, R., et al.: Findings of the 2016 conference on machine translation (wmt16). In: First Conf. on Machine Translation. pp. 131–198 (2016)
5. Chowdhery, A., Narang, S., et al.: Palm: Scaling language modeling with pathways. Journal of Machine Learning Research **24**(240), 1–113 (2023)
6. Deutsch, D., Briakou, E., et al.: WMT24++: Expanding the language coverage of WMT24 to 55 languages & dialects. In: Findings of ACL. pp. 12257–12284 (2025)
7. Elbasani, E., Kim, J.D.: Amr-cnn: Abstract meaning representation with convolution neural network for toxic content detection. Journal of Web Engineering (2022)
8. Garg, S., Galstyan, A., et al.: Extracting biomolecular interactions using semantic parsing of biomedical text. In: Proceedings of the AAAI Conference on Artificial Intelligence. vol. 30 (2016)
9. Gliwa, B., Mochol, I., et al.: SAMSum corpus: A human-annotated dialogue dataset for abstractive summarization. In: Proc. of New Frontiers in Summarization. pp. 70–79 (2019)
10. Gorrell, P., et al.: Syntax and parsing, vol. 76 (1995)
11. Jin, Z., Chen, Y., et al.: Analyzing the role of semantic representations in the era of large language models. In: Proc. of NAACL. pp. 3781–3798 (2024)
12. Kapanipathi, P., Abdelaziz, I., et al.: Leveraging Abstract Meaning Representation for knowledge base question answering. In: Findings of ACL-IJCNLP. pp. 3884–3894 (2021)
13. Kasper, R.T.: A flexible interface for linking applications to Penman's sentence generator. In: Speech and Natural Language: Proceedings of a Workshop Held at Philadelphia, Pennsylvania, February 21-23, 1989 (1989)
14. Li, B., Alkhouli, T., et al.: Eliciting better multilingual structured reasoning from LLMs through code. In: Proc. of ACL. pp. 5154–5169 (2024)
15. Liao, K., Lebanoff, L., Liu, F.: Abstract Meaning Representation for multi-document summarization. In: Proc. of COLING. pp. 1178–1190 (2018)
16. Miller, E.: An introduction to the resource description framework. D-lib Magazine (1998)
17. OpenAI: Gpt-4o mini. https://platform.openai.com (2024)
18. OpenAI: Gpt-4.1 nano. https://openai.com/index/gpt-4-1/ (2025)
19. OpenAI: gpt-oss-20b. https://openai.com/index/gpt-oss-model-card/ (2025)
20. Raut, A., Zhu, X., Pacheco, M.L.: Can LLMs interpret and leverage structured linguistic representations? a case study with AMRs. In: Proc. of XLLM. pp. 173–185 (2025)
21. Rei, R., Guerreiro, N.M., et al.: Tower+: Bridging generality and translation specialization in multilingual llms. arXiv preprint arXiv:2506.17080 (2025)
22. Shi, F., Chen, X., et al.: Large language models can be easily distracted by irrelevant context. In: International Conference on Machine Learning. pp. 31210–31227. PMLR (2023)
23. Team, G., IBM: Granite-3.3-2b. https://huggingface.co/ibm-granite/granite-3.3-2b-base (2025)

24. Team, Q.: Qwen2.5: A party of foundation models (2024), https://qwenlm.github.io/blog/qwen2.5/
25. Touvron, H., Lavril, T., et al.: Llama: Open and efficient foundation language models. arXiv preprint arXiv:2302.13971 (2023)
26. Wang, X., Li, S., Ji, H.: Code4Struct: Code generation for few-shot event structure prediction. In: Proc. of ACL. pp. 3640–3663 (2023)
27. Wein, S., Schneider, N.: Lost in translationese? reducing translation effect using Abstract Meaning Representation. pp. 753–765 (2024)
28. Yang, B., Zhao, K., et al.: Emphasising structured information: Integrating abstract meaning representation into llms for enhanced open-domain dialogue evaluation. arXiv preprint arXiv:2404.01129 (2024)
29. Yang, K., Liu, J., et al.: If llm is the wizard, then code is the wand: A survey on how code empowers large language models to serve as intelligent agents. arXiv preprint arXiv:2401.00812 (2024)
30. Zhang, J., Wang, T., et al.: SR-LLM: Rethinking the structured representation in large language model. In: Proc. of ACL. pp. 3443–3462 (2025)
31. Zhang, X., Zhao, J., LeCun, Y.: Character-level convolutional networks for text classification. Advances in neural information processing systems **28** (2015)
32. Zhang, Y., Baldridge, J., He, L.: PAWS: Paraphrase Adversaries from Word Scrambling. In: Proc. of NAACL (2019)