Prompting Decoder-Only LLMs for AMR Parsing: A Small-Scale Study on AMR 3.0

Veronika Smilga, 6766485
University of Tübingen
smilgaveronika@gmail.com

Abstract

I study whether modern decoder-only large language models (LLMs) can be prompted to produce competitive Abstract Meaning Representation parses on AMR 3.0 without task-specific fine-tuning. Using a fixed subset of 50 sentences from AMR 3.0 test set, I compare naïve zero-/few-shot prompting, retrieval-augmented (RAG) few-shot prompting with exemplar selection via sentence embeddings and FAISS, and a small self-correcting agent implemented in LangChain. I evaluate OpenAI `gpt-4o` and `gpt-5` against a strong supervised baseline (BART fine-tuned on AMR 3.0). Smatch++ is used for scoring with both full-set (invalid parses counted as zero) and valid-only metrics. The RAG-augmented pipeline demonstrates the largest single-step gains over naïve prompting. The self-correcting agent with `gpt-5` attains prefect validity and the best full-set F1 score (76.58 full-set F1, 100% validity). Nevertheless, the fine-tuned BART baseline remains stronger overall (83.51 full-set F1, 100% validity). Cost analysis[1] shows that `gpt-5`-based reasoning agent achieves higher accuracy but leads to substantially higher costs, while RAG-based approach offers a more balanced cost-performance trade-off. I conclude that while careful prompting plus lightweight agentic frameworks can substantially improve the LLMs' performance as AMR parsers compared to a naïve prompting setting, fine-tuned encoder-decoder models currently remain unrivaled. The project code is available at `https://github.com/smilni/amr-llm-parsing`.

1 Introduction

Abstract Meaning Representation (AMR) encodes sentential meaning as a rooted, directed, labeled graph with nodes as (mostly) predicate senses and edges as semantic roles (Banarescu et al., 2013). Over the last decade, AMR parsing has moved from hybrid, feature-rich pipelines to neural encoder-decoder architectures. In particular, state of the art results on AMR 2.0/3.0 are achieved by graph-aware transformers and ensemble approaches. There is growing interest in whether large decoder-only LLMs can be used out-of-the-box to deliver high-quality AMR parses without task-specific training.

Prior work has shown that in-context prompting of general-purpose LLMs often yields ill-formed or semantically off-target graphs, lagging behind specialized parsers (e.g., Ettinger et al., 2023; Li and Fowlie, 2025). Fine-tuning decoder-only models has been demonstrated to yield much better results (Ho, 2025), but this approach sacrifices the deployability and generality that make prompting appealing in the first place. Given rapid LLM improvements, it seems reasonable to reassess what various prompting strategies can achieve for AMR parsing.

I conduct a small-scale, controlled study on AMR 3.0 using a fixed 50-sentence subset to answer the following questions:

1. **RQ1: How far can naïve prompting go?** I test several zero- and few-shot prompting strategies for `gpt-4o` and `gpt-5`.

2. **RQ2: Does retrieval help more than adding more examples?** I build a RAG-augmented pipeline with top-k sentence-level neighbors retrieved at inference stage for each test sentence.

3. **RQ3: Can a self-correcting agentic loop improve the validity and accuracy of generated AMRs?** I develop a five-stage agent that iteratively drafts AMRs, checks their well-formedness via PENMAN, reflects on the drafts, and revises them according to the reflection until a valid graph is produced.

[1] My experiments were run under OpenAI's initial $20 credit for new users; cost values correspond to nominal API prices.

4. **RQ4: What approach is the best in terms of cost-efficiency?** I conduct a cost-performance analysis, plotting performance in terms of F1 vs. costs for each model and prompting setup described above.

I find that adding in-context examples improves over zero-shot for both models. However, very long instruction+exemplar prompts likely suffer from long-context effects and prove to be less effective than concise 10-shot prompts. Replacing random exemplars with retrieved neighbors yields the largest single-step gain, and a simple self-correcting agent manages to achieve 100% validity and the best LLM score (however, by a small margin). Cost-performance analysis shows that as a reasoning model, `gpt-5` is markedly more expensive than non-reasoning `gpt-4o` due to billed "thinking" tokens. Compared to a reasoning agent, RAG setups offer better cost efficiency at the cost of sometimes producing ill-formed AMR graphs. Importantly, the fine-tuned BART baseline leads overall, both due to its performance (83.51 F1, 100% validity) and cost-efficiency (free-to-use after fine-tuning).

The paper is structured as follows: in §2, I review the related work and provide motivation for using LLMs in AMR parsing; in §3, I describe the selected datasets and evaluation metrics as well as prompting setups; in §4, I report and analyze the results in terms of performance and cost trade-offs; in §5, I provide an overview of the results and outline the directions for future work.

2 Related Work

Research on AMR parsing has progressed rapidly over the past decade, moving from hybrid statistical/rule-based systems to neural encoder-decoder architectures and, more recently, to large decoder-only language models. A systematic comparison across approaches is, however, complicated by the fact that reported results are based on different benchmark versions (AMR 1.0, 2.0, and 3.0), which are not directly comparable. For more information on AMR 3.0 and how it relates to the earlier versions (1.0 and 2.0), refer to Subsection 3.1. All scores in this section are reported on test sets.

2.1 Hybrid Approaches

Early AMR parsers combined probabilistic and rule-based approaches, often explicitly encoding linguistic knowledge with the use of hand-crafted rules or sophisticated pre-processing methods.

The very first automated AMR-parsing system, JAMR (Flanigan et al., 2014), was a two-stage pipeline with plenty of hand-engineered features and normalization rules. In the concept identification stage, a linear-chain Conditional Random Field (CRF) aligned spans of text with candidate AMR concepts using lexical and syntactic features. In the relation identification stage, a log-linear edge model scored possible semantic relations, and a maximum spanning connected subgraph algorithm selected the highest-scoring, well-formed AMR graph. JAMR achieved Smatch F1 score of 58.2 on AMR 1.0.

Wang et al. (2015a,b) introduced CAMR, a transition-based parser that transformed a dependency tree into an AMR graph through a fixed inventory of hand-crafted actions, such as MERGE, LEFT-EDGE, RIGHT-EDGE, REENTRANCE. To apply these rules, the authors trained a maximum entropy classifier, selecting the most probable action at each step with the use of features from the dependency parse, lexical cues, and the partially constructed graph. CAMR achieved 66.51 Smatch F1, a performance gain of more than 7 absolute points over JAMR.

In parallel, other early work explored grammar-based approaches to AMR parsing. Artzi et al. (2015); Misra and Artzi (2016) adapted Combinatory Categorial Grammar (CCG) for AMR, tying AMR fragments to syntactic categories. These grammar-driven parsers were able to capture regularities in the syntax-semantics interface but suffered from coverage gaps and sparsity, since grammar extraction depends heavily on alignments and parallel data. CCG approach achieved 66.3 Smatch F1 on AMR 1.0.

Pust et al. (2015) proposed an AMR parser grounded in Syntax-Based Machine Translation (SBMT), treating AMR parsing as a string-to-tree transduction problem similar to translation. They transformed AMR graphs into tree structures, devised a custom AMR-targeted language model, and incorporated semantic features into the pipeline. SBMT-based parser achieved 67.7 Smatch F1 on AMR 1.0.

2.2 Neural Encoder-Decoder Models

The rise of neural encoder-decoder architectures led to a significant performance increase in AMR

parsing. Early neural parsers treated AMR graphs as linearized sequences, enabling sequence-to-sequence learning from sentences to AMR strings. For instance, unsupervised data augmentation combined with LSTM-based seq2seq models (Konstas et al., 2017) demonstrated Smatch F1 of 62.2 - a competitive result, but still below what statistical approaches described in Section 2.1 offered.

Another leap in performance occurred with transformer-based models pre-trained on large text corpora. Bevilacqua et al. (2021) introduced SPRING, a BART-based seq2seq model that handles text-to-AMR and AMR-to-text via graph linearization. SPRING reached 83.8 Smatch F1 on AMR 2.0 (84.3 with silver augmentation) and 83.0 Smatch F1 on AMR 3.0. Subsequent models suggested various enhancements for the encoder-decoder transformers with: incorporating graph structural pre-training, ancestor information, or semantic role features. Lee et al. (2023) adapted FLAN-T5, an encoder-decoder instruction-fine-tuned model, using a combination of full fine-tuning and parameter-efficient LoRA post-tuning. Their best configuration achieved 86.4 Smatch F1 on AMR 2.0 and 84.9 on AMR 3.0, setting a new state of the art at the time and outperforming prior BART-based systems such as SPRING.

At the same time, models that more directly capture graph structure were developed. Graph-based neural parsers generate AMRs via incremental graph construction or predicted relation edges, often using pointer networks or transition-based decoding to maintain well-formed graphs. Notably, Cai and Lam (2020) and Zhou et al. (2021) proposed stack- or action-based transformers that build the graph node by node, improving alignment between text tokens and AMR concepts. Cai and Lam (2020) reported a 80.2 Smatch F1 on AMR 2.0 and 75.4 Smatch F1 on AMR 1.0. Zhou et al. (2021) outperformed that with 84.3 Smatch F1 on AMR 2.0 with a single model and 84.9 Smatch F1 on AMR 2.0 with silver data and ensemble decoding.

Today's top-performing systems often ensemble multiple neural parsers or use self-distillation. The Graphene parser exemplifies this trend: by ensembling and Maximum Bayes Smatch distillation of multiple models (Lee et al., 2022), Graphene reached 85.4 Smatch F1 on the AMR 3.0 benchmark, which is the current reported state-of-the-art result, surpassing the reported quality of human annotations[2].

In summary, over the past eight years, there has been a shift from explicit symbolic rules to data-driven neural approaches. Nowadays, encoder-decoder seq2seq and graph-structured parsers dominate the field. At the same time, Opitz and Frank (2022) warn that high Smatch-scores demonstrated by neural parsers can be misleading - small but semantically significant errors, such as omitting predicates or altering crucial relation, crucially distort the meaning despite yielding a decent Smatch score. They also demonstrated that some parsers with slightly lower scores exhibit more semantically faithful outputs. The authors recommend complementing Smatch with macro-level statistics, supplementary metrics, and focused human evaluations to better capture real semantic accuracy.

2.3 Decoder-Only Language Models

An emerging paradigm is the use of large decoder-only language models (LLMs) for AMR parsing. In this case, the input sentence and the output AMR are handled in a single generative sequence. Initial studies reported that in-context prompting of OpenAI models of GPT family without fine-tuning yielded subpar results with ill-formed or incomplete graphs.

In-context prompting approach initially yielded disappointing results. Ettinger et al. (2023) showed that even with few-shot prompting, models mostly failed to produce accurate parses. None of the parses in their evaluation were acceptable - despite reasonable syntactic formatting, underlying semantics often differed drastically from gold standard graphs. In (Li and Fowlie, 2025), even with carefully crafted few-shot and chain-of-thought prompts, GPT-4 achieved Smatch F1 of only 60 on AMR 3.0 - comparable to the JAMR parser of 2014 and far below dedicated modern neural parsers. These deficiencies can be partly explained by the models' lack of structured decoding constraints - their outputs often require heavy post-processing to ensure well-formed AMRs (valid variable scoping, role labels, reentrancies).

Fine-tuning, on the other hand, demonstrated promising results. Ho (2025) showed that a fine-tuned LLaMA 3.2 model reaches a Smatch of 80

[2]As demonstrated by Banarescu et al. (2013) and pointed out by Opitz and Frank (2022), average human annotator vs. consensus inter-annotator-agreement (measured by Smatch F1) is 83 for newswire and 79 for web text.

on AMR 3.0 - on par with a strong transformer-based parser from IBM (APT with silver data) and slightly below the SOTA Graphene ensemble. This result suggested when decoder-only LLMs are directly adapted to the task via fine-tuning, they can approach the performance of specialized encoder-decoder models. At the same time, these models still struggle with maintaining valid AMR format.

2.4 Why revisit LLMs?

Overall, while decoder-only models used in an in-context learning or fine-tuning paradigm have not yet surpassed encoder-decoder parsers trained on dedicated task data, the results seem promising. Ho (2025) mentions exploring prompting strategies with closed API models like ChatGPT as a possible future research direction. Although prior work has shown that zero-shot or few-shot prompting of such models often leads to ill-formed or semantically inaccurate parses (Ettinger et al., 2023; Li and Fowlie, 2025), the continuous scaling and improvement of the models make it worthwhile to revisit this direction, which is the main goal of the current study.

3 Methods

3.1 Datasets

AMR 3.0 (Linguistic Data Consortium, 2020) is the third major release of the Abstract Meaning Representation corpus. It is the the current benchmark in the literature, which will allow us to compare the obtained results to modern neural parsers. From AMR 1.0 (Linguistic Data Consortium, 2014) to AMR 2.0 (Linguistic Data Consortium, 2017), as well as from AMR 2.0 to AMR 3.0, the dataset was expanded and the annotations refined, with more consistent treatment of PropBank frames, named entities, modality, and quantities. Thus, AMR 1.0 and 2.0 can be treated as subsets of AMR 3.0, although some instances were re-annotated in between releases. AMR 3.0 contains approximately 59255 sentence-AMR pairs across domains such as news, web text, discussion forums, Wikipedia, Aesop's Fables, and LORELEI reports, split into 55k/1.7k/1.7k train/dev/test sets. We use the version of the dataset that is publicly available on HuggingFace[3].

As using large closed-source LLMs is expensive, the experiments are conducted on a subset of 50 sentence-AMR pairs from AMR 3.0 sampled via `pandas.DataFrame.sample` method with the random seed of 42. The same subset is used in all experiments. Using a smaller dataset for the initial evaluations is common practice - for example, Ettinger et al. (2023) analyze just 30 sentences to probe error types and semantic fidelity of LLM semantic parses generation.

3.2 Evaluation metrics

Smatch (Cai and Knight, 2013) has long been the standard metric for evaluating AMR parsing. It provides a simple precision, recall, and F1 score by aligning the variables and calculating the overlap between predicted and gold-standard graphs. However, several important shortcomings of vanilla Smatch have been pointed out by Opitz and Frank (2022). Most importantly, as Smatch treats all edges equally, semantically crucial errors like dropping a predicate, reversing polarity, or misassigning a core semantic role are under-penalized. On the opposite, if a parser chooses the wrong root concept, nearly all of its outgoing relations will be considered erroneous, disproportionately lowering the score. As a result, two AMRs that differ in meaning in important ways can nonetheless receive similar Smatch values.

In spite of its shortcomings, Smatch remains the evaluation standard used in the vast majority state-of-the-art works on neural AMR parsing (Lam et al., 2022; Lee et al., 2023, i.a.). One suggested improvement that is compatible with the original idea is Smatch++ (Opitz, 2023). This package preserves the idea of the original metric, while improving on its implementation and providing more detailed evaluation feedback. In particular, Smatch++ standardizes graph pre-processing to ensure consistency, introduces lossless graph compression that enables optimal graph alignments, and provides fine-grained subscores in addition to the basic F1, precision, and recall. The resulting feedback is more reliable and interpretable, but still directly comparable to the scores obtained from other Smatch implementations. In my experiments, I use the publicly available Python package `smatchpp`[4] published by the authors of the paper.

3.3 Traditional methods

For a baseline, I rely on the `amrlib`[5] model `model_parse_xfm_bart_large`, a BART large

[3] https://huggingface.co/datasets/hoshuhan/amr-3-parsed

[4] https://github.com/flipz357/smatchpp

[5] https://github.com/bjascob/amrlib

model[6] fine-tuned on the train set of AMR 3.0. The model achieves 83.7 Smatch on AMR 3.0 test set. While this score is slightly below the current state of the art (85.4 achieved by Graphene), this model is accessible, reproducible, and does not require custom training or orchestration, unlike other approaches. That is why I consider it a practical baseline to evaluate against.

3.4 Model selection

I select models following the LMArena (Chiang et al., 2024) leaderboard[7]. LMArena ranks systems by human pairwise preferences via crowdsourcing, in a way similar to how Elo scores are computed for chess. To be able to try out a general-purpose instruction-fine-tuned model to a model with reasoning capabilities, I choose `chatgpt-4o-latest-20250326`[8] and `gpt-5`[9] respectively - ranked #2 and #1 in the overall Text categories as of August 25th, 2025. In the Appendix A, I also include results for `o3-2025-04-16`[10] - ranked #2 in the overall Text categories as of August 25th, 2025 - that I was trying out as a reasoning model before `gpt-5` was released. For the reasoning models, `max_completion_tokens` is set to `5000` to exclude the possibility of the model being stuck while reasoning, which would increase the costs and processing times.

I restrict my experiments to OpenAI models for two main reasons. First, they are among the few top-ranked systems on the LMArena leaderboard that are consistently accessible through a stable public API, making them easy to integrate. Second, focusing on a single provider helps me avoid problems caused by differences in API behavior and rate limits and use one generalized pipeline for querying each model. Trying out leading models from other providers, such as Anthropic's `claude-opus-4-1-2025-08-05-thinking` and Google's `gemini-2.5-pro` remains one of the directions for future research.

3.5 LLM prompting methods

To get structured output in AMR format from LLMs, I initially try out five different approaches in two prompting regimes - a general (zero-shot) setup, few-shot in-context prompting, and and a self-correcting draft-critique-revise agent. Every prompt's text is available in the project's GitHub repository. The same format of the user message was used in every set-up: "`Generate an Abstract Meaning Representation (AMR) graph for the following sentence: {sentence} AMR parse:`".

3.5.1 Naïve prompting (zero-/few-shot)

The most straightforward approach to prompting is a general zero-shot setup. In that case, I specify the task and desired output format in natural language without task examples, relying solely on the model's existing knowledge to produce the graphs.

A slightly more sophisticated approach is few-shot prompting. In this case, the model conditions on the examples given in the prompt without any gradient updates or fine-tuning (Brown et al., 2020; Liu et al., 2021). I try out two different approaches to few-shot prompting. First, I use 10 random labeled exemplars from the train set in the user message. Second, I create a detailed system prompt with 247 example pairs by manually adapting the instructions from AMR 1.2.6 Specification[11].

3.5.2 RAG-based few-shot prompting

Building on the results achieved in Subsection 3.5.1, I implement a simple RAG pipeline to select 10 most relevant labeled exemplars from the train set instead of supplying random ones.

For the RAG pipeline, I use `SentenceTransformer` encoder (with `BAAI/bge-base-en-v1.5`, a lightweight, retrieval-tuned encoder with competitive performance on English data). Vectors and metadata are cached and indexed via FAISS `IndexFlatIP` (exact inner-product, equivalent to cosine similarity on normalized vectors). At query time, I perform top-k search to obtain similarity scores. Top-10 sentences with highest similarity scores are mapped back to their paired gold AMR parses and provided to the LLM in the user message at inference. To speed up index construction and retrieval over this corpus, all embedding-related computations were run on Google Colab GPUs (a single T4 GPU node).

[6] https://huggingface.co/facebook/bart-large

[7] https://lmarena.ai/leaderboard

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

[9] https://platform.openai.com/docs/models/gpt-5

[10] https://platform.openai.com/docs/models/o3

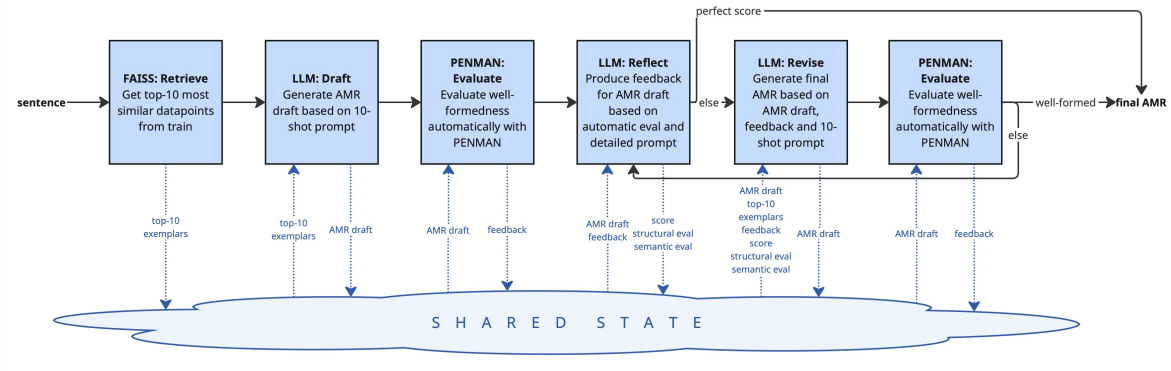


Figure 1: Pipeline of the self-correcting AMR-parsing agent.

3.5.3 Self-correcting agent

This approach is inspired by Self-Refine (Madaan et al., 2023) and Reflexion (Shinn et al., 2023) frameworks and implemented in LangChain. The resulting agent is a small state machine with five nodes (Retrieve → Draft → Evaluate → Reflect → Revise) connected by conditional routing. The pipeline is presented in Figure 1.

Given a sentence, top-k (k=10) exemplars are retrieved from the train set via a `FAISS` index over `BAAI/bge-base-en-v1.5` embeddings and pass these to the Draft node. The Evaluate node runs an automatic structural check on the output of the Draft node (PENMAN well-formedness + brief diagnostics) and writes both the draft and feedback to a shared state. The Reflect node prompts an LLM-critic to produce a concise score and error notes conditioned on the auto-feedback and an extended rule/exemplar prompt.

If the score is 5/5, the process is terminated; otherwise the Revise node generates a corrected AMR using the draft, feedback, and exemplars. The revised AMR graph is re-evaluated by the Evaluate node. If well-formed, the graph is returned as the final AMR. Otherwise, the agent loops back to the Reflect node for a correction pass. The loop is repeated until a well-formed AMR graph is obtained.

4 Results

4.1 Performance

Table 2 reports Smatch metrics computed on the subset of valid AMRs. 95% CIs from a paired bootstrap over sentences (N=50) provided by Smatch++ package are reported in Appendix B. I report full-set scores (invalid outputs counted as zero/empty) as well as the valid-only ones (invalid outputs disregarded).

4.1.1 Naïve prompting (zero-/few-shot)

Across models, adding in-context examples improves quality over zero-shot. For `gpt-5`, F1 rises from 67.83 (0-shot) to 68.59 (10-shot) and 70.82 (247-shot), with validity stable at 92-96%. `gpt-4o` attains perfect validity in 0-shot (100%) but lower completeness (55.54 F1); in a few-shot setting, it achieves F1 scores of 60.57 (10-shot) and 63.44 (247-shot) with validity of 96% and 90%, respectively. Overall, the best performance in this category is achieved by the newest model (`gpt-5`) and the most detailed prompt (247-shot) of those considered.

4.1.2 RAG-based few-shot prompting

Replacing random exemplars with retrieved most similar ones yields the largest single-step gains. `gpt-5` improves to 76.02 F1 (98% valid), a +5.20 gain over its 247-shot prompt (most successful in the naïve setting), while `gpt-4o` reaches 69.12 (+5.68; 92% valid). These gains come from balanced increases in precision and recall and indicate that exemplar relevance matters more than exemplar count, with just 10 relevant sentence–AMR pairs yielding better performance than detailed instructions and 247 general examples. Additionally, I speculate that the detailed instruction (18,000+ tokens) may harm the models' performance due to well-known long-context effects, such as recency bias, attention dilution, and pattern interference. Important rules introduced early in the prompt can be effectively "lost and forgotten" at the start of generation. In contrast, a short set of retrieved, relevant exemplars placed near the query improves the model's performance greatly.

[11] https://github.com/amrisi/amr-guidelines/blob/master/amr.md

Model	Prompting setup	Prompt tokens	Completion tokens	Total tokens	Cost / request (USD)
gpt-4o	instruction, 0-shot	154.9 ± 10.97	118.82 ± 70.82	273.72 ± 80.87	0.00158 ± 0.00073
gpt-4o	instruction, 10-shot	1358.9 ± 10.97	126.48 ± 76.98	1485.38 ± 87.16	0.00466 ± 0.00080
gpt-4o	detailed instr., 247-shot	13059.9 ± 10.97	121.58 ± 68.08	13181.48 ± 78.12	0.03387 ± 0.00071
gpt-4o	RAG 10-shot	2009.92 ± 925.17	135.76 ± 80.38	2145.68 ± 989.18	0.00638 ± 0.00298
gpt-4o	LangGraph agent	19227.9 ± 3350.31	312.06 ± 201.84	19539.96 ± 3532.24	0.05119 ± 0.01022
gpt-5	instruction, 0-shot	153.9 ± 10.97	2859.66 ± 1400.28	3013.56 ± 1409.00	0.02879 ± 0.01401
gpt-5	instruction, 10-shot	1357.9 ± 10.97	3110.84 ± 1539.52	4468.74 ± 1548.65	0.03281 ± 0.01541
gpt-5	detailed instr., 247-shot	12955.9 ± 10.97	3088.12 ± 1637.22	16044.02 ± 1646.27	0.04708 ± 0.01638
gpt-5	RAG 10-shot	2008.92 ± 925.17	2463.06 ± 1366.90	4471.98 ± 1969.32	0.02714 ± 0.01423
gpt-5	LangGraph agent	18672.84 ± 2980.44	6589.18 ± 3729.37	25262.02 ± 6198.31	0.08923 ± 0.04000

Table 1: Per-request token usage and cost (mean $\pm$ std). Costs are computed using gpt-4o ($2.50/M input tokens, $10/M output tokens) and gpt-5 ($1.25/M input tokens, $10/M output tokens) pricing.

Model	Prompting setup	Valid (%)	F1	F1	Precision	Recall
			Full-set	Valid-only	Valid-only	Valid-only
BART (baseline)	fine-tuned (train)	**100.00**	**83.51**	**83.51**	**81.60**	**85.51**
gpt-4o	instruction, 0-shot	**100.00**	55.54	55.54	50.41	61.83
gpt-4o	instruction, 10-shot	96.00	58.15	60.57	58.17	63.17
gpt-4o	detailed instr., 247-shot	90.00	57.10	63.44	59.15	68.41
gpt-4o	RAG 10-shot	92.00	63.59	69.12	67.72	70.57
gpt-4o	LangGraph agent	100.00	66.32	66.32	62.10	71.16
gpt-5	instruction, 0-shot	92.00	62.40	67.83	65.49	70.34
gpt-5	instruction, 10-shot	96.00	65.85	68.59	66.87	70.41
gpt-5	detailed instr., 247-shot	96.00	67.98	70.82	68.68	73.10
gpt-5	RAG 10-shot	98.00	74.50	76.02	75.69	76.35
gpt-5	LangGraph agent	**100.00**	***76.58***	***76.58***	***76.56***	***76.61***

Table 2: AMR parsing results on a subset of 50 sentences from AMR 3.0 test set. Full-set F1 score is calculated on the full set of generated AMR parses, with invalid outputs treated as 0. Valid-only F1/Precision/Recall are calculated on the subset of valid AMR parses only.

4.1.3 Self-correcting agent

The Retrieve $\rightarrow$ Draft $\rightarrow$ Evaluate $\rightarrow$ Reflect $\rightarrow$ Revise agent achieves 100% validity for both gpt-5 and gpt-4o thanks to iterative AMR graph refinement that continues until the PENMAN well-formedness checks are passed. For gpt-5, it also slightly outperforms the RAG-only approach with a full-set F1 of 76.58 vs. 76.02 (+0.56) and a valid-only F1 of 76.58 vs. 74.50 (+2.08). For gpt-4o, full-set F1 improves as well due to the increase in validity percentage with 63.59 vs. 66.32 (+2.73), but valid-only F1 for the reasoning agent fails to surpass that of the RAG-only approach with 66.32 vs. 69.12 (-2.80). Fixing structural issues (which can lead to improved validity) does not imply fixing semantic problems, and it seems that gpt-4o-based agent failed to do so.

4.2 Cost considerations

Table 1 summarizes per-request token usage and corresponding costs for each model-prompting configuration. Notably, using gpt-5 for AMR parsing turns out to be more expensive than using gpt-4o in spite of a lower base price ($1.25/M input tokens, $10/M output tokens for gpt-5 vs. $2.50/M input tokens, $10/M output tokens for gpt-4o). The difference is explained by the fact that, unlike gpt-4o, gpt-5 is a reasoning model, with reasoning tokens are billed as completion tokens[12]. That is reflected in Table 1, which shows comparable numbers of prompt tokens for the two models and a sharp, more than 20x increase in completion tokens for gpt-5 compared to gpt-4o.

Figure 2 visualizes the same data as a cost-performance trade-off, with full-set F1 scores plotted against mean cost per request. The gray curve marks the Pareto-optimal configurations, i.e. those that achieve the highest F1 for a given cost. Thus, this line is the efficiency frontier across prompting setups: points on the curve correspond to a cost-efficient improvement in quality, while being below the line signals diminishing performance gains rel-

[12] According to https://platform.openai.com/docs/guides/reasoning.

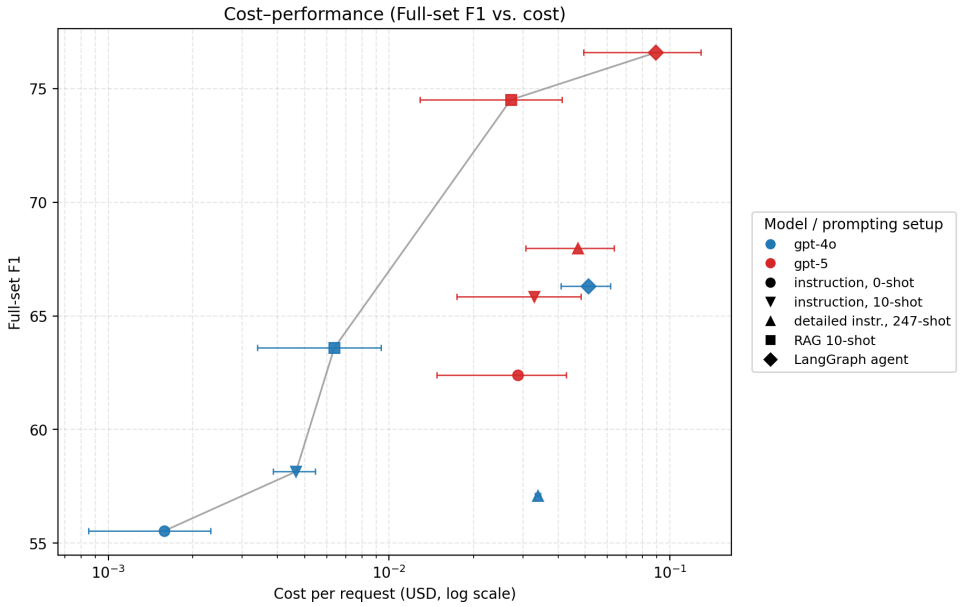


Figure 2: Cost-performance trade-off across models and prompting setups. Each point shows the mean cost per request (log-scaled, with horizontal bars denoting one standard deviation) against the corresponding full-set Smatch F1 score. The gray line connects non-dominated (Pareto-optimal) configurations.

ative to the increase in cost.

While `gpt-5` generally achieves higher accuracy at all spending levels, this comes with a steep increase in cost. `gpt-5`-based reasoning agent reaches the highest Smatch F1 of 76.58, but it is at the same time the most expensive configuration, making it the best choice for settings where accuracy is prioritized over cost-efficiency. Mid-range setups such as `gpt-5` / `gpt-4o` with retrieval-augmented prompting (RAG 10-shot) offer more balanced cost-effectiveness. At the lowest-cost regime (≈0.001-0.007, USD per request), `gpt-4o` with zero- or few-shot prompting achieves reasonable accuracy of ≈56-64 F1, making it the most suitable choice for large-scale or exploratory runs.

4.3 Discussion

`gpt-5` shows a significant improvement in performance compared to `gpt-4o`, + ≈7-11 full-set F1 percentage points when evaluated in the same setting and with the same prompts. More sophisticated prompting setups consistently lead to performance gains for both models, with greatest improvement in full-set F1 demonstrated by RAG-augmented pipeline compared to random 10-shot, +5.14 and +8.65 for `gpt-4o` and `gpt-5` respectively. Multi-step reasoning agent does not yield consistent performance gains in terms of F1, but allows to ensure 100% validity of generated AMRs. However, fine-tuned BART baseline remains strongest overall: 100% validity and 83.51 F1, exceeding the best LLM prompting result of 100% validity and 76.58 F1 achieved by the `gpt-5`-based agent (an almost 7 percentage points difference).

In addition to the numerical results, I also looked at several examples of good and bad AMR parses. An in-depth analysis of several parses can be found in Appendix C. The best setup, LangGraph agent with `gpt-5`, produced structurally correct graphs in all cases and mostly captured the meaning well. The analyzed poor (in terms of F1) outputs came from inconsistent gold annotations or small differences in how meanings can be represented – for example, using `say-01` instead of `state-01`. In some cases the model's version was not wrong, but more direct or simplified.

It is important to note that earlier work (Banarescu et al., 2013) has shown that even human annotators reach only about 80 Smatch F1 agreement, limiting how accurately any system can be evaluated. To address this, future work may need to revisit evaluation standards or at least use multiple gold parses per sentence to capture acceptable variation.

Let us now revisit the four research questions introduced in Section 1 and relate them to the empirical results reported above.

RQ1: How far can naïve prompting go? Zero- and few-shot prompting of decoder-only LLMs

produces usable AMRs with Smatch F1 comparable to that of early 2010s' approaches described in Section 2.1. However, its performance lags much behind modern neural parsers. Few-shot improves over zero-shot for both `gpt-4o` and `gpt-5`. However, detailed instruction+exemplar prompts (247-shot) perform worse than concise 10-shot prompts in terms of full-set F1. Overall, naïve prompting is viable for exploratory runs but cannot competite with the fine-tuned baseline.

RQ2: Does retrieval help more than adding more examples? Yes. Replacing random exemplars with retrieved neighbors yields the largest single-step gains: with only 10 retrieved sentence–AMR pairs, `gpt-5` improves to 76.02 full-set F1 (+5.20 over its best naïve prompt) and `gpt-4o` to 63.59 full-set F1 (+5.14 over 10-shot). Exemplar relevance matters more than exemplar count.

RQ3: Can a self-correcting agent improve validity and accuracy? Yes, more so for validity than for accuracy. The Retrieve→Draft→Evaluate→Reflect→Revise agent achieves 100% validity with both models as a backbone. It yields a modest additional F1 gain over RAG for `gpt-5` (76.58 vs. 76.02; +0.56 full-set), but not for `gpt-4o`. Thus, iterative self-correction guarantees validity, but accuracy benefits are limited and model-dependent.

RQ4: What approach is best if cost-efficiency is factored in? Crucially, the fine-tuned BART baseline remains strongest overall (83.51 F1, 100% validity) and has near-zero marginal cost after training. Out of LLM-based approaches, `gpt-5` is the most accurate, but also the most expensive due to the costs of reasoning tokens. The best setup in terms of cost–performance trade-off is RAG 10-shot (for either model), while the lowest-cost regime is `gpt-4o` zero-/few-shot with ≈56–64 F1.

5 Conclusion

This small-scale study on the 50-sentence subset of AMR 3.0 shows that careful prompting can push decoder-only LLMs to produce usable AMR parses, but a supervised encoder–decoder BART baseline still wins by a large margin (83.51 Smatch F1 with 100% validity vs. 76.58 Smatch F1 with 100% validity). This gap is not unexpected: BART has been fine-tuned on ∼55,000 training instances from AMR 3.0, while LLM-based approaches operate in zero-/few-shot regimes without access to full training data (even for the RAG pipeline, the model only has direct access to 10 most relevant examples from the training set).

Among prompting strategies, retrieval-augmented few-shot (10 retrieved exemplars) demonstrates the largest single-step gain over naïve prompting, and a simple self-correcting agent reliably ensures AMR well-formedness. Cost–performance analysis reveals consistent trade-offs: while `gpt-5` generally yields higher accuracy than `gpt-4o` at comparable prompting setups, it is also more expensive in practice despite a lower base price per input token due to the billing of reasoning tokens as completion tokens. On the Pareto frontier, retrieval-augmented prompting setups occupy the "sweet spot", while the reasoning agent achieves a small boost in F1 and guaranteed validity at the highest cost.

There are several important limitations to the current study. First, the test set is limited (50 sentences out of 1.700 in the AMR 3.0 test set). Second, only two closed-source backbone models (`gpt-4o` and `gpt-5`) are evaluated. Third, for evaluation I only report Smatch++ (full-set and valid-only) without human semantic judgments. Finally, confidence intervals are calculated automatically by Smatch++ and reflect sentence resampling rather than multiple stochastic runs, not addressing the inherent variability of LLM responses.

Another important observation concerns the evaluation setup. The strongest fine-tuned neural parsers may achieve very high Smatch scores partly because they reproduce patterns and annotation biases from the training data rather than capture true semantic content. As human inter-annotator agreement for AMR parsing rarely exceeds about 80 F1, some variation between valid graphs is permissible and even expected and desirable. Conversely, current evaluation standards seem to favor surface similarity. A more reliable assessment would require multiple gold parses per sentence and/or human evaluation of the parser generations complementary to the automatically calculated Smatch scores.

Directions for future work include addressing the limitations and expanding the scope of the current study. First, evaluation should be performed on larger and more diverse subsets of AMR 3.0 (or the full test set), ideally with human assessments to complement automatic Smatch metrics. Second, additional backbone models should be evaluated (e.g., state-of-the-art reasoning and non-reasoning

models from multiple providers, not limited to OpenAI). That may include lightweight fine-tuning (e.g., with LoRA) of smaller open-weight decoder-only models as a middle ground between prompting and fine-tuning, similarly to that performed in Ho (2025). Finally, it may be interesting to investigate hybrid pipelines that combine supervised parsers for structure with LLMs for targeted semantic repairs in case of failure.

References

Yoav Artzi, Kenton Lee, and Luke Zettlemoyer. 2015. Broad-coverage CCG semantic parsing with AMR. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*, pages 1699–1710, Lisbon, Portugal. Association for Computational Linguistics.

Laura Banarescu, Claire Bonial, Shu Cai, Madalina Georgescu, Kira Griffitt, Ulf Hermjakob, Kevin Knight, Philipp Koehn, Martha Palmer, and Nathan Schneider. 2013. Abstract Meaning Representation for sembanking. In *Proceedings of the 7th Linguistic Annotation Workshop and Interoperability with Discourse*, pages 178–186, Sofia, Bulgaria. Association for Computational Linguistics.

Michele Bevilacqua, Rexhina Blloshmi, and Roberto Navigli. 2021. One spring to rule them both: Symmetric amr semantic parsing and generation without a complex pipeline. In *Proceedings of the AAAI conference on artificial intelligence*, volume 35, pages 12564–12573.

Tom B. Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel M. Ziegler, Jeffrey Wu, Clemens Winter, Christopher Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. 2020. Language models are few-shot learners.

Deng Cai and Wai Lam. 2020. AMR parsing via graph-sequence iterative inference. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 1290–1301, Online. Association for Computational Linguistics.

Shu Cai and Kevin Knight. 2013. Smatch: an evaluation metric for semantic feature structures. In *Proceedings of the 51st Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, pages 748–752, Sofia, Bulgaria. Association for Computational Linguistics.

Wei-Lin Chiang, Lianmin Zheng, Ying Sheng, Anastasios Nikolas Angelopoulos, Tianle Li, Dacheng Li, Hao Zhang, Banghua Zhu, Michael Jordan, Joseph E. Gonzalez, and Ion Stoica. 2024. Chatbot arena: An open platform for evaluating llms by human preference.

Allyson Ettinger, Jena Hwang, Valentina Pyatkin, Chandra Bhagavatula, and Yejin Choi. 2023. "you are an expert linguistic annotator": Limits of LLMs as analyzers of Abstract Meaning Representation. In *Findings of the Association for Computational Linguistics: EMNLP 2023*, pages 8250–8263, Singapore. Association for Computational Linguistics.

Jeffrey Flanigan, Sam Thomson, Jaime Carbonell, Chris Dyer, and Noah A. Smith. 2014. A discriminative graph-based parser for the Abstract Meaning Representation. In *Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1426–1436, Baltimore, Maryland. Association for Computational Linguistics.

Shu Han Ho. 2025. Evaluation of finetuned llms in amr parsing.

Ioannis Konstas, Srinivasan Iyer, Mark Yatskar, Yejin Choi, and Luke Zettlemoyer. 2017. Neural AMR: Sequence-to-sequence models for parsing and generation. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 146–157, Vancouver, Canada. Association for Computational Linguistics.

Hoang Thanh Lam, Gabriele Picco, Yufang Hou, Young-Suk Lee, Lam M. Nguyen, Dzung T. Phan, Vanessa López, and Ramon Fernandez Astudillo. 2022. Ensembling graph predictions for amr parsing.

Young-Suk Lee, Ramón Astudillo, Hoang Thanh Lam, Tahira Naseem, Radu Florian, and Salim Roukos. 2022. Maximum Bayes Smatch ensemble distillation for AMR parsing. In *Proceedings of the 2022 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 5379–5392, Seattle, United States. Association for Computational Linguistics.

Young-Suk Lee, Ramón Fernandez Astudillo, Radu Florian, Tahira Naseem, and Salim Roukos. 2023. Amr parsing with instruction fine-tuned pre-trained language models.

Yanming Li and Meaghan Fowlie. 2025. Gpt makes a poor amr parser. *Journal for Language Technology and Computational Linguistics*, 38(2):43–76.

Linguistic Data Consortium. 2014. Abstract meaning representation (amr) 1.0. `https://catalog.ldc.upenn.edu/LDC2014T12`. Accessed on August 15, 2025.

Linguistic Data Consortium. 2017. Abstract meaning representation (amr) 2.0. `https://catalog.ldc.upenn.edu/LDC2017T10`. Accessed on August 15, 2025.

Linguistic Data Consortium. 2020. Abstract meaning representation (amr) 3.0. https://catalog.ldc.upenn.edu/LDC2020T02. Accessed on August 15, 2025.

Pengfei Liu, Weizhe Yuan, Jinlan Fu, Zhengbao Jiang, Hiroaki Hayashi, and Graham Neubig. 2021. Pre-train, prompt, and predict: A systematic survey of prompting methods in natural language processing.

Aman Madaan, Niket Tandon, Prakhar Gupta, Skyler Hallinan, Luyu Gao, Sarah Wiegreffe, Uri Alon, Nouha Dziri, Shrimai Prabhumoye, Yiming Yang, Shashank Gupta, Bodhisattwa Prasad Majumder, Katherine Hermann, Sean Welleck, Amir Yazdanbakhsh, and Peter Clark. 2023. Self-refine: Iterative refinement with self-feedback.

Dipendra Kumar Misra and Yoav Artzi. 2016. Neural shift-reduce CCG semantic parsing. In *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*, pages 1775–1786, Austin, Texas. Association for Computational Linguistics.

Juri Opitz. 2023. SMATCH++: Standardized and extended evaluation of semantic graphs. In *Findings of the Association for Computational Linguistics: EACL 2023*, pages 1595–1607, Dubrovnik, Croatia. Association for Computational Linguistics.

Juri Opitz and Anette Frank. 2022. Better Smatch = better parser? AMR evaluation is not so simple anymore. In *Proceedings of the 3rd Workshop on Evaluation and Comparison of NLP Systems*, pages 32–43, Online. Association for Computational Linguistics.

Michael Pust, Ulf Hermjakob, Kevin Knight, Daniel Marcu, and Jonathan May. 2015. Parsing English into Abstract Meaning Representation using syntax-based machine translation. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*, pages 1143–1154, Lisbon, Portugal. Association for Computational Linguistics.

Noah Shinn, Federico Cassano, Edward Berman, Ashwin Gopinath, Karthik Narasimhan, and Shunyu Yao. 2023. Reflexion: Language agents with verbal reinforcement learning.

Chuan Wang, Nianwen Xue, and Sameer Pradhan. 2015a. Boosting transition-based AMR parsing with refined actions and auxiliary analyzers. In *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing (Volume 2: Short Papers)*, pages 857–862, Beijing, China. Association for Computational Linguistics.

Chuan Wang, Nianwen Xue, and Sameer Pradhan. 2015b. A transition-based algorithm for AMR parsing. In *Proceedings of the 2015 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 366–375, Denver, Colorado. Association for Computational Linguistics.

Jiawei Zhou, Tahira Naseem, Ramón Fernandez Astudillo, Young-Suk Lee, Radu Florian, and Salim Roukos. 2021. Structure-aware fine-tuning of sequence-to-sequence transformers for transition-based AMR parsing. In *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*, pages 6279–6290, Online and Punta Cana, Dominican Republic. Association for Computational Linguistics.

A Results for `o3-2025-04-16` with Confidence Intervals

Due to cost considerations, I did not run any experiments with `o3-2025-04-16` after I switched to a newly released `gpt-5`. Thus, `o3-2025-04-16` was not tested as a backbone for the LangChain agent. The costs for `o3-2025-04-16` were also not recorded.

Model	Prompting setup	Valid AMRs (%)	F1 [95% CI]	Precision [95% CI]	Recall [95% CI]
o3	instruction, 0-shot	92.00	61.40 [58.19, 64.88]	57.06 [53.26, 61.01]	66.45 [63.14, 69.77]
o3	instruction, 10-shot	86.00	66.42 [62.16, 70.55]	64.66 [60.11, 69.45]	68.28 [63.44, 72.29]
o3	detailed instr., 247-shot	94.00	70.37 [66.34, 74.14]	67.14 [62.32, 71.66]	73.93 [70.16, 77.35]
o3	RAG 10-shot	96.00	72.80 [68.35, 77.18]	71.48 [66.90, 76.17]	74.16 [69.28, 78.63]

Table 3: AMR parsing results for `o3-2025-04-16`. All metrics are computed on the subset of *valid* AMRs only; brackets denote 95% confidence intervals as reported by Smatch++.

B Results for `gpt-5` and `gpt-4o` with Confidence Intervals

Paired bootstrap resamples sentences with replacement and recomputes metrics to reflect test-sample variability. It does not capture generation stochasticity that is inevitable with LLMs with temperature set to larger than 0. Therefore, in this case CIs should be treated as indicative and not definitive.

Model	**Prompting setup**	**Valid (%)**	**F1 [95% CI]**	**Precision [95% CI]**	**Recall [95% CI]**
BART (baseline)	fine-tuned (train)	**100.00**	**83.51 [79.29, 86.90]**	**81.60 [77.98, 84.78]**	**85.51 [80.19, 89.36]**
`gpt-4o`	instruction, 0-shot	**100.00**	55.54 [51.45, 59.61]	50.41 [46.24, 54.51]	61.83 [57.50, 66.00]
`gpt-4o`	instruction, 10-shot	96.00	60.57 [56.41, 64.35]	58.17 [53.74, 62.53]	63.17 [58.19, 66.74]
`gpt-4o`	detailed instr., 247-shot	90.00	63.44 [59.59, 67.14]	59.15 [54.90, 63.37]	68.41 [64.61, 71.77]
`gpt-4o`	RAG 10-shot	92.00	69.12 [64.25, 73.88]	67.72 [62.48, 72.74]	70.57 [65.52, 75.46]
`gpt-4o`	LangGraph agent	**100.00**	66.32 [61.20, 71.36]	62.10 [56.04, 67.94]	71.16 [66.69, 75.68]
`gpt-5`	instruction, 0-shot	92.00	67.83 [63.35, 71.47]	65.49 [60.85, 69.37]	70.34 [65.19, 74.36]
`gpt-5`	instruction, 10-shot	96.00	68.59 [64.68, 72.05]	66.87 [62.53, 70.80]	70.41 [66.17, 74.03]
`gpt-5`	detailed instr., 247-shot	96.00	70.82 [66.45, 75.03]	68.68 [63.68, 73.45]	73.10 [68.54, 77.03]
`gpt-5`	RAG 10-shot	98.00	76.02 [71.50, 80.14]	75.69 [70.51, 80.10]	76.35 [71.39, 80.64]
`gpt-5`	LangGraph agent	**100.00**	***76.58 [72.55, 80.69]***	***76.56 [72.46, 80.60]***	***76.61 [71.89, 80.90]***

Table 4: AMR parsing results for `gpt-5` and `gpt-4o`, plus a supervised fine-tuned baseline. All metrics are computed on the subset of *valid* AMRs only; brackets denote 95% confidence intervals as reported by Smatch++.

C Qualitative analysis of AMR Examples

In this section, I analyze in detail several graphs generated by the best-performing setup, i.e. LangGraph Agent with `gpt-5` as its backbone model. All graph visualizations were generated with the use of `amrlib.graph_processing.amr_plot.AMRPlot`.

C.1 Lowest F1 AMR Examples

1. m1456.

Gold AMR Generated AMR

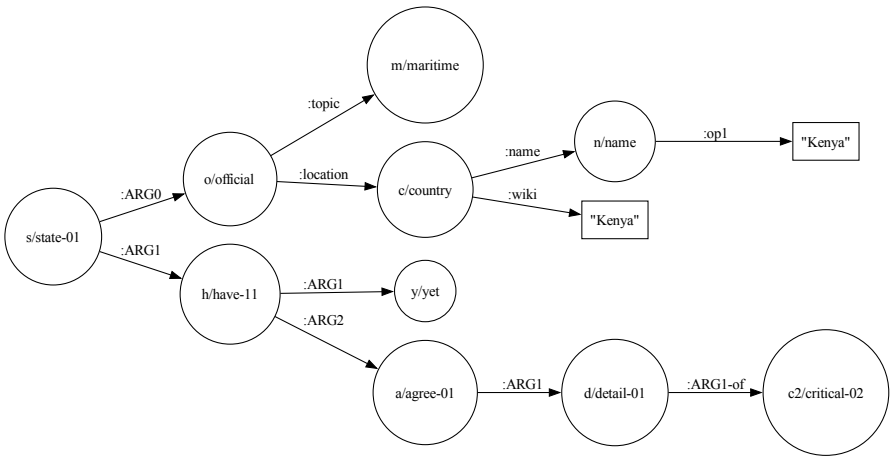
Figure 3: Gold vs. generated AMR. Smatch++ F1: 0.0

For the non-linguistic input "m1456.", the gold AMR represents a string literal (`string-entity :value "m1456"`), while the model output is an empty graph (`amr-empty`). This yields an F1 score of 0.0, a complete structural mismatch.

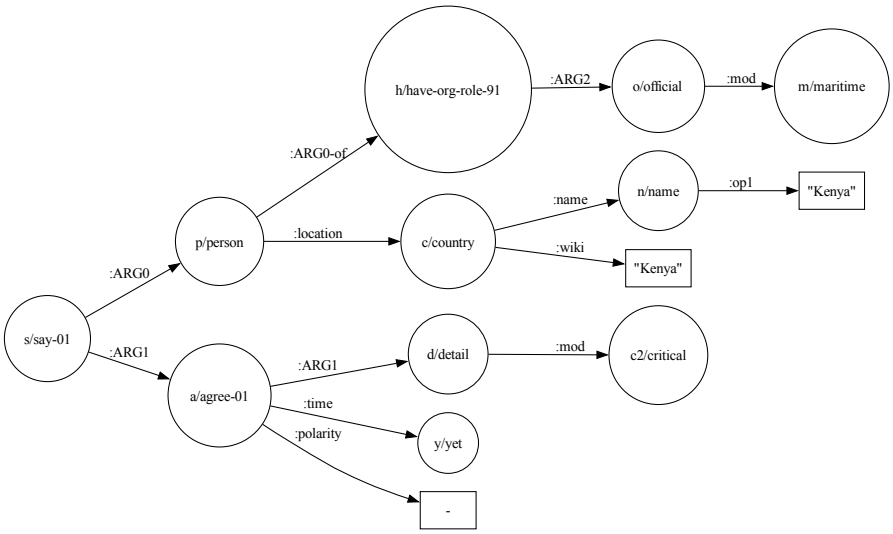
I would argue that in this case the the flaws of AMR 3.0 annotation lead to parsing errors. When examining the train set, I found several cases with similar non-linguistic strings and varying gold parses. In one case, the gold AMR for `m1954` was (`s / string-entity :value "m1954"`), similar to the gold AMR for the test datapoint in question. However, in another case, the gold AMR for `m1470` was (`a / amr-empty`), same as the Agent's output.

The LangGraph Agent makes use of most similar examples from train to generate its outputs. Conflicting examples make it difficult for the parser to learn a consistent mapping. Thus, the Agent's empty output in this case may reflect the inconsistent annotation in AMR 3.0 rather than be a parsing failure.

2. Maritime officials in Kenya stated that critical details have yet to be agreed upon.



Gold AMR



Generated AMR

Figure 4: Gold vs. generated AMR. Smatch++ F1: 46.81

The gold AMR uses `state-01` to represent the reporting event and encodes the agent as `official` with the topic `maritime` and location `Kenya`, while the generated AMR replaces it with `say-01` and expands the agent as `person` using `have-org-role-91` to represent `official` with modifier `maritime`. Both choices are consistent with the AMR design principles. However, according to the AMR guidelines, the gold relation `:topic` is more semantically precise than the generated `:mod` as `maritime` is a domain of expertise rather than a simple descriptive modifier. A similar difference appears in the treatment of `critical details`: the gold AMR represents `critical` as a predicative adjective (`detail-01 :ARG1-of critical-02`), following the convention that adjectives should be reified as predicates taking their modified entity as `:ARG1`, while the generated AMR simplifies it to `detail :mod critical`.

The main difference between the graphs is how “have yet to be agreed upon" is represented: the gold graph introduces a light-verb structure `have-11`, while the generated AMR simplifies the construction by

making `agree-01` the central predicate, attaching `yet` directly to it and marking `:polarity -`.

Overall, both graphs convey the same meaning but differ in structure and abstraction level. The gold AMR stays closer to the surface syntax, while the generated AMR represents the semantics more directly by adding explicit negation, simplifying auxiliary verbs, and reducing adjectival structures. This makes the generated version more transparent but slightly less aligned with standard AMR conventions that aim to preserve syntactic framing. Still, neither graph contains major structural flaws, and the differences between generated and gold graphs seem to be within normal variation in AMR realizations[13], unlike the issues demonstrated by earlier neural parsers and discussed in Opitz and Frank (2022) (see Subsection 2.2 for details on that).

[13] Of course, to prove that claim, I would need to collect human annotations for the generated parses, which is out of the scope of this project.

C.2 Highest F1 AMR Examples

1. 2008-10-08

Gold AMR Generated AMR

Figure 5: Gold vs. generated AMR. Smatch++ F1: 100.0

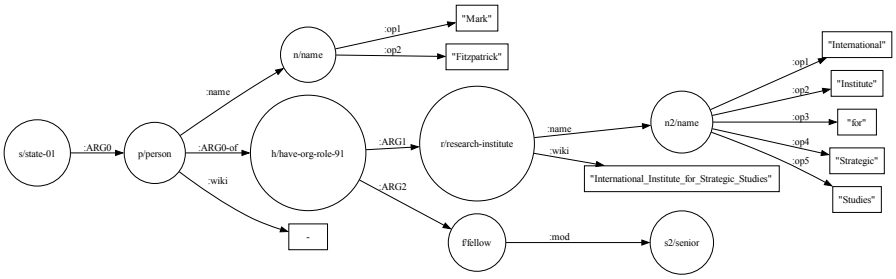
2. Crime; weapons; international; money

Gold AMR Generated AMR

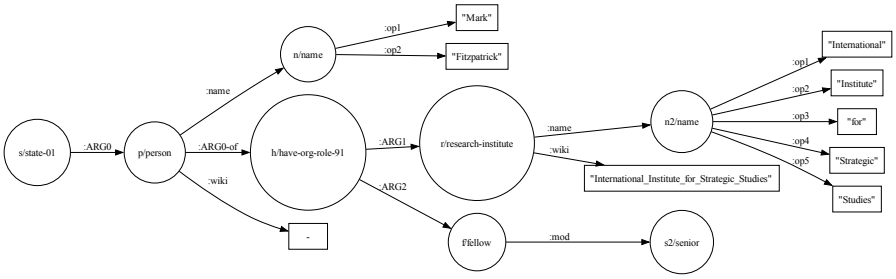
Figure 6: Gold vs. generated AMR. Smatch++ F1: 100.0

These examples of graphs for semantically primitive and highly compositional inputs show a perfect match between gold and generated AMRs, with the F1 of 100.0. The model correctly represents dates and lists using the standard AMR structures `date-entity` and `and`. This suggests that it handles simple, structured inputs reliably – accurately reproducing the gold structure.

3. Senior Fellow at the International Institute for Strategic Studies mark Fitzpatrick stated that –



Gold AMR

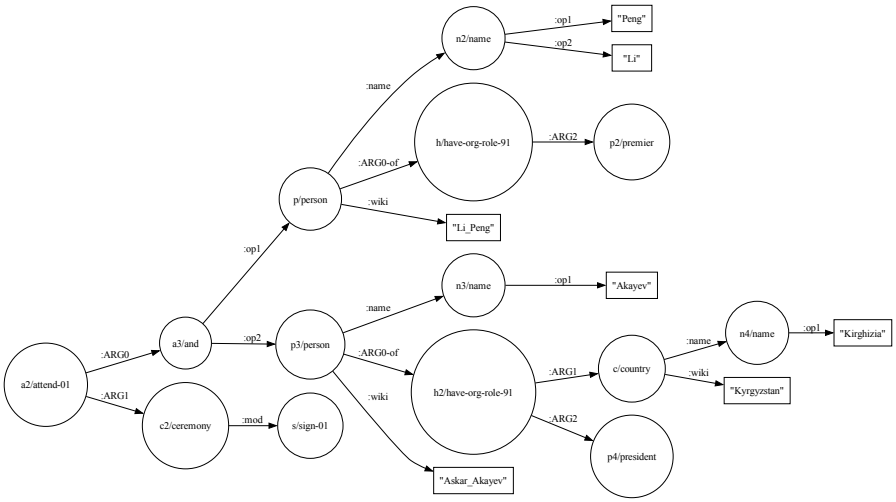


Generated AMR

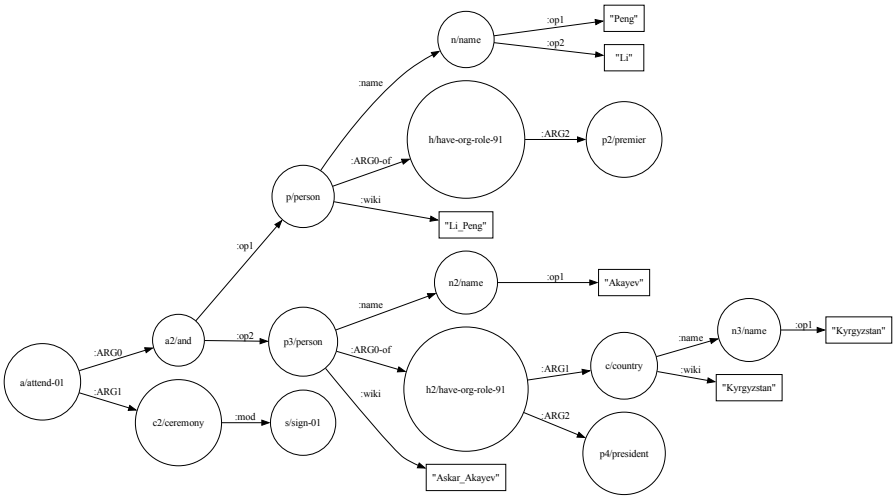
Figure 7: Gold vs. generated AMR. Smatch++ F1: 100.0

This example is more complex than the previous ones, with predicate structure (`state-01`) rather than a simple date or list. The graph correctly captures the agent's professional title and affiliation through the frame `have-org-role-91`, linking `person`, `fellow`, and `research-institute` as well as modifier `senior` for `fellow`. It also encodes the nested organization name "International Institute for Strategic Studies" with the appropriate `name` and `wiki` attributes. This example shows that the parser can handle more complex sentences with hierarchical roles and predicate-argument structure.

4. Premier Peng Li and Kirghizian President Akayev attended the signing ceremony .



Gold AMR



Generated AMR

Figure 8: Gold vs. generated AMR. Smatch++ F1: 97.14

This example involves a more complex coordination structure, with two agents and embedded role frames. Both AMRs correctly represent the joint event "attended the signing ceremony," using `attend-01` and `and` to link the two participants. Each agent's title and affiliation are modeled through `have-org-role-91` with correct `name`, `country`, and `wiki` attributes. The generated AMR is almost identical to the gold one, differing only in the location name (`Kirghizia` vs. `Kyrgyzstan`, with the first one being closer to the original sentence and the second one being the correct country name in English).