SAFT: Structure-Aware Fine-Tuning of LLMs for AMR-to-Text Generation

Rafiq Kamel[*,1,†] **Filippo Guerranti**[*,1,2] **Simon Geisler**[1,2,‡] **Stephan Günnemann**[1,2]

[1]School of Computation, Information and Technology, Technical University of Munich, Germany
[2]Munich Data Science Institute, Technical University of Munich, Germany
{r.kamel,f.guerranti,s.geisler,s.guennemann}@tum.de

Abstract

Large Language Models (LLMs) are increasingly applied to tasks involving structured inputs such as graphs. Abstract Meaning Representations (AMRs), which encode rich semantics as directed graphs, offer a rigorous testbed for evaluating LLMs on text generation from such structures. Yet, current methods often arbitrarily linearize AMRs, discarding key structural cues, or rely on architectures incompatible with standard LLMs. We introduce SAFT, a structure-aware fine-tuning approach that injects graph topology into pretrained LLMs without architectural changes. We compute direction-sensitive positional encodings from the magnetic Laplacian of transformed AMRs and project them into the embedding space of the LLM. While possibly applicable to any graph-structured inputs, we focus on AMR-to-text generation as a representative and challenging benchmark. SAFT sets a new state-of-the-art on AMR 3.0 with a 3.5 BLEU improvement over baselines. Gains scale with graph complexity, highlighting the value of structure-aware representations in enhancing LLM performance. SAFT offers a general and effective pathway for bridging structured data and language models.

1 Introduction

Large Language Models (LLMs) have become the dominant paradigm for natural language processing (NLP), demonstrating strong generalization across a wide range of sequential tasks. Increasingly, researchers are exploring how to extend the capabilities of LLMs to structured data domains such as graphs (Jin et al., 2024a; Jiang et al., 2023; Fatemi et al., 2024; Zhang et al., 2022; Tang et al., 2024), driven by growing interest in extending the reasoning and representation capabilities of LLMs beyond sequential data to more expressive, structured modalities. However, existing approaches that adapt LLMs to graphs often require architectural modifications, or auxiliary components. These strategies compromise a core advantage of LLMs: their scalability and flexibility as pretrained, general-purpose sequence models.

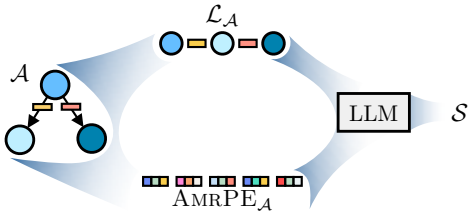


Figure 1: **SAFT**: structure-aware fine-tuning of LLMs for AMR-to-text ($\mathcal{A} \rightarrow \mathcal{S}$), using linearized AMRs ($\mathcal{L}_{\mathcal{A}}$) augmented with AMR-specific PEs (AMRPE$_{\mathcal{A}}$).

A particularly well-defined and linguistically grounded graph representation is the Abstract Meaning Representation (AMR) (Banarescu et al., 2013), a rooted, directed acyclic graph that encodes

[*]Equal contribution.
[†]Now at Technical University of Nuremberg.
[‡]Now at Google.

Accepted at the KDD2025 Workshop on Structured Knowledge for LLMs.

predicate-argument structure and core semantic relations. We focus on the AMR-to-text generation task: producing a natural language sentence that accurately expresses the meaning of an AMR graph. This task presents a strong benchmark for evaluating the ability of LLMs to interface with structured semantic representations, as it demands sensitivity to graph topology and semantic content while preserving fluency and coherence in the generated output.

Despite its importance, AMR-to-text generation remains challenging due to the inherent relational and semantical structure of AMRs. Sequence-to-sequence models (Bevilacqua et al., 2021; Cheng et al., 2022) linearize AMRs, discarding structural information crucial for semantic fidelity. Graph-to-sequence methods (Song et al., 2018; Zhu et al., 2019; Ribeiro et al., 2021) preserve structure through Graph Neural Networks (GNNs), but their reliance on specialized encoders breaks compatibility with pretrained LLMs. More recent work attempts to repurpose LLMs for this task via prompting or fine-tuning on linearized AMRs (Mager et al., 2020; Yao et al., 2024a; Raut et al., 2025), but these methods still overlook the underlying graph structure critical to meaning preservation. This fragmentation reveals a critical gap:

How can graph-structured information be integrated into LLMs in a lightweight, architecture-agnostic way to enable structure-aware generation?

We address this question with **SAFT**, a structure-aware fine-tuning method that augments LLM inputs with positional encodings derived from graph topology. Specifically, we compute direction-sensitive graph positional encodings from the magnetic Laplacian (Furutani et al., 2020; Geisler et al., 2023) of an AMR-derived graph and inject them into the embeddings of the graph linearization tokens via a lightweight projection network. This design ensures compatibility with any decoder-only LLM and avoids architectural changes to the model, as illustrated in Fig. 1.

While our approach is grounded in AMR-to-text generation, its design is conceptually applicable to other tasks involving graph-structured inputs, such as drug design (Zheng et al., 2024), code representation learning (Allamanis et al., 2017), and scene graph-to-text generation (Yang et al., 2019). We focus on AMRs as they provide a linguistically motivated, semantically rich benchmark that allows for precise evaluation of structural understanding in language generation. Their formalism enables controlled experimentation with topology and meaning, making them an ideal foundation for this line of work. SAFT provides a concrete step toward aligning structured graph representations with pretrained LLMs, focusing on AMRs as a high-value benchmark for studying structure-aware generation.

Our contributions include:

- A **structure-aware fine-tuning framework for LLMs** that incorporates graph positional encodings into token embeddings, enabling relational inductive bias without modifying the model architecture.
- A novel formulation of **AMR-specific positional encodings** derived from the eigenvectors of the magnetic Laplacian, effectively capturing directionality and global semantic structure in AMRs.
- Comprehensive experiments showing that SAFT achieves **state-of-the-art performance** on AMR 3.0, with a +3.5 BLEU improvement over baselines, and increased gains on graphs with higher structural complexity, such as document-level AMRs.

2 Background

We introduce the foundational concepts necessary for our method, focusing on graph representations and graph positional encodings, Abstract Meaning Representations (AMRs), and the application of Large Language Models (LLMs) to structured input.

2.1 Graph Representations and Graph Positional Encodings

Edge-labeled directed graphs. AMRs are a prime example of edge-labeled directed graphs. We formally define these as tuples $\mathcal{A} = (\mathcal{V}_\mathcal{A}, \mathcal{E}_\mathcal{A}, \mathcal{R}_\mathcal{A}) \in \mathbb{A}$, where $\mathcal{V}_\mathcal{A}$ is the set of $n_\mathcal{A} = |\mathcal{V}_\mathcal{A}|$ nodes, $\mathcal{E}_\mathcal{A} \subseteq \mathcal{V}_\mathcal{A} \times \mathcal{V}_\mathcal{A}$ is the set of $m_\mathcal{A} = |\mathcal{E}_\mathcal{A}|$ directed edges, and $\mathcal{R}_\mathcal{A}$ is a finite set of relation types (edge labels), such that each edge $(u, v) \in \mathcal{E}_\mathcal{A}$ is associated with a label $r_{u,v} \in \mathcal{R}_\mathcal{A}$. The space of these graphs is denoted by $\mathbb{A}$.

Edge-unlabeled directed graphs. To understand positional encodings on graphs, it is useful to first consider edge-unlabeled directed graphs, defined as tuples $\mathcal{G} = (\mathcal{V}_\mathcal{G}, \mathcal{E}_\mathcal{G}) \in \mathbb{G}$, where $\mathcal{V}_\mathcal{G}$ is the set of $n_\mathcal{G} = |\mathcal{V}_\mathcal{G}|$ nodes, and $\mathcal{E}_\mathcal{G} \subseteq \mathcal{V}_\mathcal{G} \times \mathcal{V}_\mathcal{G}$ is the set of $m_\mathcal{G} = |\mathcal{E}_\mathcal{G}|$ directed, binary, and unlabeled edges, with $e_{u,v} = 1$ if there is a directed edge from node u to node v, and 0 otherwise. The space of such graphs is denoted by $\mathbb{G}$. The relational structure is encoded in the adjacency matrix $\boldsymbol{A} \in \{0,1\}^{n_\mathcal{G} \times n_\mathcal{G}}$, where $\boldsymbol{A}_{u,v} = e_{u,v}$. We define the out-degree matrix $\boldsymbol{D}$ as a diagonal matrix with $\boldsymbol{D}_{u,u} = \sum_v \boldsymbol{A}_{u,v}$. Symmetrizing the adjacency matrix as $\boldsymbol{A}_S = \boldsymbol{A} \vee \boldsymbol{A}^\top$ (element-wise logical OR) yields an undirected representation of the graph, with a corresponding symmetrized degree matrix $\boldsymbol{D}_S$. This allows for the computation of the symmetric normalized Laplacian $\boldsymbol{L}_S = \boldsymbol{I} - \boldsymbol{D}_S^{-1/2} \boldsymbol{A}_S \boldsymbol{D}_S^{-1/2}$, which has a real eigendecomposition $\boldsymbol{L}_S = \boldsymbol{U} \boldsymbol{\Lambda} \boldsymbol{U}^\top$, where $\boldsymbol{U} \in \mathbb{R}^{n_\mathcal{G} \times n_\mathcal{G}}$ contains orthonormal eigenvectors and $\boldsymbol{\Lambda} = \text{diag}(\lambda_1, \ldots, \lambda_{n_\mathcal{G}}) \in \mathbb{R}^{n_\mathcal{G} \times n_\mathcal{G}}$ is a diagonal matrix of real eigenvalues. However, this symmetrization inherently loses the directional information present in the original directed graph.

Magnetic Laplacian. To address the loss of directionality, we can employ the magnetic Laplacian (Furutani et al., 2020), which introduces directional information via complex-valued phase shifts. For $q \in \mathbb{R}_{\geq 0}$, the magnetic Laplacian $\boldsymbol{L}^{(q)} \in \mathbb{C}^{n_\mathcal{G} \times n_\mathcal{G}}$ is defined as:

$$\boldsymbol{L}^{(q)} := \boldsymbol{D}_S - \boldsymbol{A}_S \odot \exp\left(i \boldsymbol{\Theta}^{(q)}\right), \tag{1}$$

where $i = \sqrt{-1}$, $\odot$ denotes the Hadamard product (element-wise multiplication), and the phase matrix $\boldsymbol{\Theta}^{(q)} \in \mathbb{R}^{n_\mathcal{G} \times n_\mathcal{G}}$ is given by $(\boldsymbol{\Theta}^{(q)})_{u,v} = 2\pi q((\boldsymbol{A})_{u,v} - (\boldsymbol{A})_{v,u})$. The magnetic Laplacian $\boldsymbol{L}^{(q)}$ is Hermitian, guaranteeing a complete set of complex eigenvectors $\boldsymbol{\Gamma} \in \mathbb{C}^{n_\mathcal{G} \times n_\mathcal{G}}$.

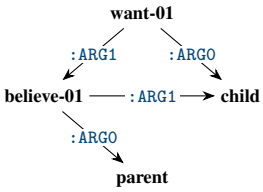
Graph Positional Encodings (GPEs) assign each node a notion of position within the graph. Most common approaches leverage the spectral properties of the graph Laplacian to encode the structural position of nodes. In particular, the eigenvectors of the Laplacian provide an orthonormal basis that captures the graph's structure at varying frequencies.

Following (Belkin and Niyogi, 2003; Dwivedi and Bresson, 2021), we can define the Laplacian-based positional encoding $\phi(v_i) \in \mathbb{R}^k$ for a node $v_i \in \mathcal{V}_\mathcal{G}$ as the i-th row of the first k eigenvectors in $\boldsymbol{U}$:

$$\phi(v_i) = [\boldsymbol{U}_{i,1}, \ldots, \boldsymbol{U}_{i,k}]^\top, \tag{2}$$

where $k \ll n_\mathcal{G}$ is a chosen dimensionality. These embeddings inherently capture coarse-to-fine structural patterns and are invariant to node permutations.

2.2 Abstract Meaning Representation



(a) Graph representation

```
(w / want-01
  :ARG0 (c / child)
  :ARG1 (b / believe-01
    :ARG0 (p / parent)
    :ARG1 c))
```

(b) Penman notation

```
<P0> want-01 :ARG0 <P1> child
:ARG1 <P2> believe-01 <stop>
<P2> :ARG0 <P3> parent :ARG1
<P1> <stop>
```

(c) BFS linearization

Figure 2: Three aligned representations of the sentence *"The child wants the parent to believe them."*: (a) a graph-based AMR structure, (b) its corresponding Penman notation, and (c) a BFS linearization used for sequence-based processing.

Abstract Meaning Representation (AMR) (Langkilde and Knight, 1998; Banarescu et al., 2013; Mansouri, 2025) is a semantic formalism that represents the meaning of a sentence as a rooted, directed acyclic graph $\mathcal{A} = (\mathcal{V}_\mathcal{A}, \mathcal{E}_\mathcal{A}, \mathcal{R}_\mathcal{A})$. The nodes $\mathcal{V}_\mathcal{A}$ represent concepts, which are typically predicates, entities, or abstract ideas. The directed edges $\mathcal{E}_\mathcal{A} \subseteq \mathcal{V}_\mathcal{A} \times \mathcal{V}_\mathcal{A}$ capture the semantic relationships between these concepts. Each edge $(u, v) \in \mathcal{E}_\mathcal{A}$ is associated with a label $r_{u,v} \in \mathcal{R}_\mathcal{A}$, where $\mathcal{R}_\mathcal{A}$ is a finite set of predefined semantic roles. Common relation labels include `:ARG0` (agent), `:ARG1` (patient/theme), and `:mod` (modifier). A key characteristic of AMR is its abstraction from surface syntax, ensuring that sentences with equivalent semantics are mapped to isomorphic AMR graphs. For instance, the sentences "*The child wants the parent to believe them*" and "*What the child wanted is for the parent to believe them*" share the same underlying AMR structure (Fig. 2a).

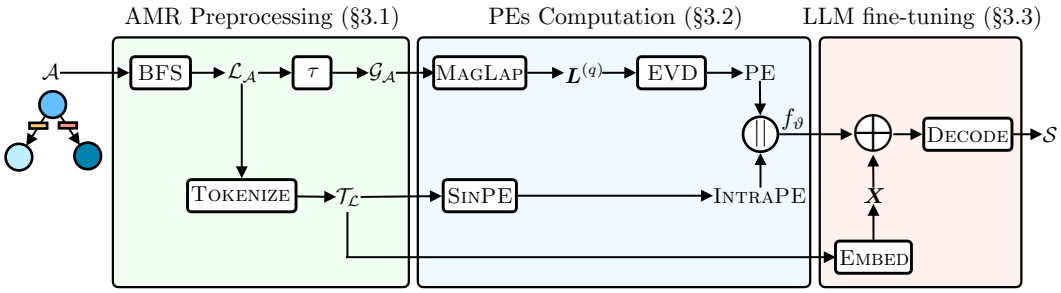


Figure 3: **Overview of SAFT.** Given the AMR linearization $\mathcal{L}_\mathcal{A}$, we construct its transformation $\mathcal{G}_\mathcal{A} = \tau(\mathcal{A})$, compute graph PEs from the magnetic Laplacian $\boldsymbol{L}^{(q)}$ of $\mathcal{G}_\mathcal{A}$, and combine them with intra-node token PEs. The result is projected via f_ϑ and injected into the token embeddings $\boldsymbol{X}$ of the tokenized linearization, enabling structure-aware generation without modifying the LLM architecture.

AMR Linearizations. To enable the processing of AMR graphs by sequence-to-sequence models, such as LLMs, it is necessary to linearize the graph structure into a sequential format. This process, termed *linearization*, transforms an AMR graph $\mathcal{A}$ into a sequence of labels $\mathcal{L}_\mathcal{A} = (\ell_1, \ell_2, \ldots, \ell_L) \in \Sigma^*$, where each ℓ_i is a label from a predefined vocabulary Σ. A common serialization is the Penman notation (Kasper, 1989; Bateman, 1990; Goodman, 2020) (Fig. 2b), a parenthetical representation that encodes the graph's concepts and relations in a compact textual form.

More recently, methods like breadth-first search (BFS) and depth-first search (DFS) based linearizations have been developed (Bevilacqua et al., 2021). For example, BFS linearization traverses the graph level by level, employing special tokens to denote relation types and reentrancies, resulting in a structured sequence that aims at preserving the graph's information for autoregressive learning (Konstas et al., 2017) (Fig. 2c). Additional details and visualizations are provided in Appendix A.

2.3 Blueprint of Large Language Models

Large Language Models (LLMs) (Vaswani et al., 2017; Devlin et al., 2019; Brown et al., 2020; Touvron et al., 2023) are parameterized functions $\pi_\theta : \Lambda^* \to \Delta^{|\Lambda|^m - 1}$ mapping an input token sequence $x \in \Lambda^*$ to a probability distribution over output sequences $y \in \Lambda^m$ of length m. Here, θ denotes the model parameters, Λ the token vocabulary, and $\Delta^{|\Lambda|^m - 1}$ the probability simplex over $\mathbb{R}^{|\Lambda|^m}$. Outputs are generated autoregressively via $\pi_\theta(y \mid x) = \prod_{t=1}^{m} \pi_\theta(y_t \mid y_{<t}, x)$, with $y_{<t} = (y_1, \ldots, y_{t-1})$.

LLMs are typically pretrained on massive text corpora by predicting the next token in a sequence. This enables them to learn intricate linguistic and semantic patterns, resulting in strong generalization capabilities across various natural language processing tasks, often without task-specific supervision. To adapt a pretrained LLM to a specific downstream task, its parameters θ are fine-tuned on a task-specific dataset $\mathcal{D} = \{(x^{(i)}, y^{(i)})\}_{i=1}^{N}$, where $x^{(i)} \in \Lambda^*$ and $y^{(i)} \in \Lambda^m$.

3 Structure-Aware Fine-Tuning for AMR-to-Text Generation

We present SAFT, a lightweight method for fine-tuning pretrained LLMs on AMR-to-text generation by incorporating structural information from the input graph. The key idea is to inject graph positional encodings, derived from the magnetic Laplacian of the AMR graph, into the token embeddings during fine-tuning. This guides the model to better capture graph topology and long-range dependencies.

Task Definition. Given an AMR graph $\mathcal{A} \in \mathbb{A}$, the goal is to generate a natural language sentence $\mathcal{S} \in \Sigma^*$ such that $\mathcal{S} = \psi(\mathcal{A})$ is fluent and semantically faithful to the input.

Approach. SAFT enhances LLM decoding by conditioning on structure-aware graph representations. We first apply a semantics-preserving transformation to the AMR graph (Section 3.1), compute positional encodings from its magnetic Laplacian (Section 3.2), and inject them into the LLM's embedding space during fine-tuning (Section 3.3).

3.1 Semantically-Preserving Transformation of AMR Graphs

Edge-labeled graphs, such as AMRs, pose a challenge for computing eigenvectors of the graph Laplacian, a standard step in deriving graph positional encodings. Applying the Laplacian directly would necessitate ignoring the crucial semantic information encoded in their edge labels. To overcome this, we introduce a transformation τ that converts a linearized AMR into a directed, edge-unlabeled *semantic-preserving graph* (SPG). The SPG retains the core semantics of the original AMR structure while enabling the application of Laplacian-based spectral methods for positional encoding.

BFS Linearization. We begin by applying a breadth-first search (BFS) traversal of the AMR graph $\mathcal{A}$, yielding a linearized sequence of labels $\mathcal{L}_\mathcal{A} = (\ell_1, \ldots, \ell_L) \in \Sigma^*$, where each ℓ_i represents a concept or role label. Each label corresponds to a node in the SPG $\mathcal{G}_\mathcal{A} = (\mathcal{V}_\mathcal{G}, \mathcal{E}_\mathcal{G})$, through an injective function $\sigma_\mathcal{A} : \mathbb{Z} \rightarrowtail \mathcal{V}_\mathcal{G}$ that aligns labels to their corresponding node in the graphs, i.e, $\sigma_\mathcal{A}(i) = v_i$. This implies $|\mathcal{V}_\mathcal{G}| = L$.

SPG Transformation. The transformation $\tau : \Sigma^* \times (\Sigma \rightarrowtail \mathcal{V}_\mathcal{G}) \rightarrow \mathbb{G}$ constructs the SPG $\mathcal{G}_\mathcal{A} = \tau(\mathcal{L}_\mathcal{A}, \sigma_\mathcal{A})$ from the label sequence and alignment mapping. Labeled edges in the original AMR are represented as role nodes in the SPG, preserving role semantics via directed edges to their source and target concept nodes. We unite co-referring nodes (e.g., marked with `<P1>`), and merge their connectivity. The resulting SPG is semantically equivalent to the original AMR but uses unlabeled edges for spectral compatibility, and explicits re-entrancies and coreferences. Additional details and visualization of the semantically-preserving transformation are available in Appendix B.1.

Tokenization of Node Labels. Each textual label $\ell_i \in \mathcal{L}_\mathcal{A}$ is associated with a node $v_i \in \mathcal{V}_\mathcal{G}$, $v_i = \sigma_\mathcal{A}(i)$. We tokenize each node label into a sequence of tokens $\boldsymbol{t}^{(\ell_i)} \in \Lambda^{p_i}$

$$\boldsymbol{t}^{(\ell_i)} = \textsc{Tokenize}(\ell_i) = (t_1^{(\ell_i)}, \ldots, t_{p_i}^{(\ell_i)}), \tag{3}$$

where Λ denotes the tokenizer's vocabulary and p_i is the number of tokens produced from the label ℓ_i. When $p_i > 1$, we refer to $v_i = \sigma_\mathcal{A}(i)$ as a *multi-token node*.

3.2 AMR-Specific Positional Encodings

In the previous section we defined the transformation from an AMR graph $\mathcal{A}$ to its semantically-preserving representation $\mathcal{G}_\mathcal{A}$ that allows us to apply spectral graph theory and compute graph positional encodings. Here, we present our AMR-specific graph positional encodings, which we compute from the magnetic Laplacian (Section 2.1) of the semantic-preserving graph $\mathcal{G}_\mathcal{A}$. These encodings are meant to capture the topology of the AMR structure and its directionality.

Node-level PEs. The magnetic Laplacian, defined in Eq. (1), encodes directionality through complex phase shifts. We compute the magnetic Laplacian $\boldsymbol{L}^{(q)}$ of $\mathcal{G}_\mathcal{A}$ and extract the eigenvectors corresponding to the lowest k eigenvalues, forming a complex matrix $\boldsymbol{\Gamma} \in \mathbb{C}^{n_\mathcal{G} \times k}$. Each node $v_i \in \mathcal{V}_\mathcal{G}$ is assigned a complex-valued k-dimensional embedding $\phi(v_i) \in \mathbb{C}^k$:

$$\phi(v_i) = [\boldsymbol{\Gamma}_{i,1}, \ldots, \boldsymbol{\Gamma}_{i,k}]^\top. \tag{4}$$

We convert $\phi(v_i)$ to a real-valued vector $\text{PE}^{(v_i)} \in \mathbb{R}^{2k}$ by concatenating the real and imaginary parts:

$$\text{PE}^{(v_i)} = [\Re(\phi(v_i)) \; \Im(\phi(v_i))]. \tag{5}$$

These positional encodings provide a spectral representation that reflects both local and global graph structure. Nodes with similar structural roles in the AMR, such as arguments or modifiers, will have similar embeddings, even if distant in the graph.

Intra-node Token Positional Encodings. For each token $t_j^{(\ell_i)}$ in the tokenized label of a node $v_i = \sigma_\mathcal{A}(i)$ we apply sinusoidal positional encodings (Vaswani et al., 2017) to preserve their intra-node ordering:

$$\textsc{IntraPE}_j^{(\ell_i)} = \textsc{SinPE}(j), \quad \text{for } j = 1, \ldots, p_i, \tag{6}$$

where $\textsc{IntraPE}_j^{(\ell_i)} \in \mathbb{R}^d$. For single-token nodes ($p_i = 1$), we use $\textsc{IntraPE}_1^{(v_i)} = \textsc{SinPE}(0)$.

AMR Positional Encodings. We combine the node-level and intra-node positional encodings to obtain the final AMR-specific positional encoding for each token $t_j^{(\ell_i)}$:

$$\textsc{AmrPE}_j^{(\ell_i)} = f_\vartheta \left(\text{PE}^{(v_i)} \,\|\, \textsc{IntraPE}_j^{(\ell_i)} \right), \tag{7}$$

Algorithm 1 AMR-to-Text Generation with SAFT

Input: AMR graph $\mathcal{A}$, pretrained LLM $\pi_\theta = \textsc{Decode} \circ \textsc{Embed}$
Output: Generated text sequence $\mathcal{S}$

$\mathcal{L}_\mathcal{A}, \sigma_\mathcal{A} = \text{BFS}(\mathcal{A})$ ▷ Linearize AMR and align labels
$\mathcal{G}_\mathcal{A} = \tau(\mathcal{L}_\mathcal{A}, \sigma_\mathcal{A})$ ▷ Transform to semantic-preserving graph (SPG)
$\mathbf{\Gamma} = \textsc{MagLapEVD}(\mathcal{G}_\mathcal{A}, k)$ ▷ Compute magnetic Laplacian eigvecs
for each $(i, \ell_i) \in \text{enumerate}(\mathcal{L}_\mathcal{A})$ **do**
  $v_i = \sigma_\mathcal{A}(i)$
  $\boldsymbol{t}^{(\ell_i)} = \textsc{Tokenize}(\ell_i)$ ▷ Tokenize label (Eq. (3))
  $\phi(v_i) = \mathbf{\Gamma}^\top_{i,:}$ ▷ Select complex eigenvector (Eq. (4))
  $\text{PE}^{(v_i)} = [\Re(\phi(v_i)) \; \Im(\phi(v_i))]$ ▷ Node-level PE (Eq. (5))
  for each token $(j, t_j^{(\ell_i)}) \in \text{enumerate}(\boldsymbol{t}^{(\ell_i)})$ **do**
    $\textsc{IntraPE}_j^{(\ell_i)} = \textsc{SinPE}(j)$ ▷ Intra-node PE (Eq. (6))
    $\textsc{AmrPE}_j^{(\ell_i)} = f_\vartheta(\text{PE}^{(v_i)} \,\|\, \textsc{IntraPE}_j^{(\ell_i)})$ ▷ Token-wise AMR PE (Eq. (7))
  end for
end for
$\textsc{AmrPE} = (\textsc{AmrPE}_1^{(\ell_1)} \; \ldots \; \textsc{AmrPE}_{p_L}^{(\ell_L)})^\top$ ▷ AMR PE (Eq. (8))
$\mathcal{T}_\mathcal{L} = \boldsymbol{t}^{(\ell_1)} \,\|\, \boldsymbol{t}^{(\ell_2)} \,\|\, \ldots \,\|\, \boldsymbol{t}^{(\ell_L)}$ ▷ Token sequence (Eq. (9))
$\boldsymbol{X} = \textsc{Embed}(\mathcal{T}_\mathcal{L})$ ▷ Token sequence embedding
$\boldsymbol{H} = \boldsymbol{X} + \textsc{AmrPE}$ ▷ Inject structure-aware PE (Eq. (11))
$\mathcal{S} = \textsc{Decode}(\boldsymbol{H})$ ▷ Generate output sequence (Eq. (12))

where $\textsc{AmrPE}_j^{(\ell_i)} \in \mathbb{R}^{d_\text{emb}}$, $f_\vartheta : \mathbb{R}^{2k+d} \to \mathbb{R}^{d_\text{emb}}$ is a two-layer MLP with GeLU activation function (Hendrycks and Gimpel, 2016), and $\|$ denotes vector concatenation. This projection defines a token-level positional encodings that captures (*i*) structural knowledge from the node-level positional encodings, and (*ii*) label-level sequential information from the intra-node token positional encodings. The embedding is mapped into the LLM embedding space (d_emb, see Section 3.3), allowing seamless integration during fine-tuning.

Concatenating the positional encodings across all nodes/labels in their linearized order, as defined by $\sigma_\mathcal{A}$, determines the final AMR-specific positional encodings matrix:

$$\textsc{AmrPE} = \left(\textsc{AmrPE}_1^{(\ell_1)} \; \ldots \; \textsc{AmrPE}_{p_1}^{(\ell_1)} \; \ldots \; \textsc{AmrPE}_1^{(\ell_2)} \; \ldots \; \textsc{AmrPE}_{p_L}^{(\ell_L)}\right)^\top, \tag{8}$$

where $\textsc{AmrPE} \in \mathbb{R}^{p \times d_\text{emb}}$ and $p = \sum_{i=1}^{L} p_i$ is the total number of tokens in the linearization. AMRPE is a representation of each token in the linearization that considers both the position of the token within its node-label and the global position in the graph.

3.3 LLM Fine-Tuning with AMR-Specific Positional Encodings

For ease of exposition, we represent the pretrained LLM decoder model as a composition:

$$\pi_\theta = \textsc{Decode} \circ \textsc{Embed}$$

where $\textsc{Embed} : \Lambda^p \to \mathbb{R}^{p \times d_\text{emb}}$ maps a sequence of tokens into the LLM's embedding space, and $\textsc{Decode} : \mathbb{R}^{p \times d_\text{emb}} \to \Sigma^*$ generates the output text sequence[4].

Given an AMR graph $\mathcal{A}$, we obtain a linearized sequence of node labels $\mathcal{L}_\mathcal{A} = (\ell_1, \ldots, \ell_L)$ and their corresponding tokenized forms $\boldsymbol{t}^{(\ell_i)} = (t_1^{(i)}, \ldots, t_{p_i}^{(i)})$. The overall token sequence $\mathcal{T}_\mathcal{L} \in \Lambda^p$ is:

$$\mathcal{T}_\mathcal{L} = \boldsymbol{t}^{(\ell_1)} \,\|\, \boldsymbol{t}^{(\ell_2)} \,\|\, \ldots \,\|\, \boldsymbol{t}^{(\ell_L)} = (t_1, \ldots, t_p) \tag{9}$$

where $p = \sum_{i=1}^{L} p_i$ is the total number of tokens in the linearized graph. The sequence $\mathcal{T}_\mathcal{L}$ is mapped to the LLM embedding space as:

$$\boldsymbol{X} = \textsc{Embed}(\mathcal{T}_\mathcal{L}), \tag{10}$$

with $\boldsymbol{X} \in \mathbb{R}^{p \times d_\text{emb}}$.

[4] DECODE is an abstraction over decoder components, including the transformer layers, head, and tokenizer.

Integrating AMR positional encodings. We integrate structure-aware positional encodings to the embedded representation of the linearized sequence of tokens. The final structure-aware embeddings used for decoding are:

$$\boldsymbol{H} = \textsc{Embed}(\mathcal{T}_{\mathcal{L}}) + \textsc{AmrPE} \tag{11}$$

where $\boldsymbol{H} \in \mathbb{R}^{p \times d_{\text{emb}}}$. Finally $\boldsymbol{H}$ is then fed to the LLM decoder to return the generated output sequence $\mathcal{S} \in \Sigma^*$:

$$\mathcal{S} = \textsc{Decode}(\boldsymbol{H}). \tag{12}$$

Prompt Handling. For clarity and modularity, we exclude the prompt segment from the structure-aware positional encoding process. The prompt is tokenized independently from the AMR linearization to avoid disrupting the alignment between graph nodes and tokens. Its tokens are embedded using the standard learned embeddings without any additional positional encodings beyond those already handled by the pretrained model. This design simplifies the architecture and ensures that the inductive bias introduced by our positional encodings applies exclusively to the AMR portion of the input. Additional details are provided in Appendix B.

4 Experiments

We evaluate our structure-aware fine-tuning approach on the state-of-the-art dataset AMR 3.0. We show that SAFT sets a new state of the art for sentence-level AMR-to-text generation (Section 4.2), and that it surpasses conventionally fine-tuned LLMs, particularly as input graph complexity increases (Section 4.3). Finally, we push graph complexity to its limits by examining document-level AMRs, revealing even more significant performance gains for SAFT (Section 4.4).

4.1 Experimental Setup

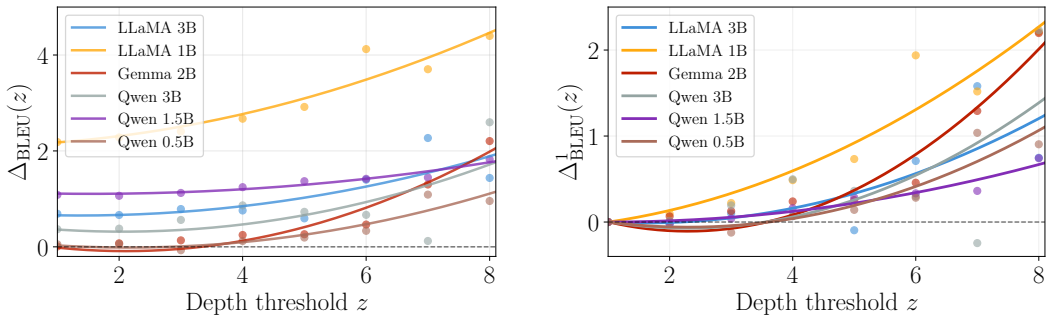
We use AMR 3.0 (LDC2020T02) (Knight et al., 2020) and DocAMR, which incorporates discourse structure and inter-sentence dependencies; split details are provided in Appendix E. We fine-tune several pretrained LLMs using Low-Rank Adaptation (LoRA) (Hu et al., 2022), including LLaMA 3.2 (1B, 3B) (Touvron et al., 2023), Qwen 2.5 (0.5B, 1.5B, 3B) (Bai et al., 2023), and Gemma (2B) (Team et al., 2024), each evaluated with and without our graph-based positional encoding module (SAFT) using the best-performing hyperparameters. We report BLEU (Papineni et al., 2002) and chrF++ (Popović, 2015) scores using greedy decoding for generation. All experiments are implemented with the `LitGPT` framework; further training and hardware details, including hyperparameters, are presented in Appendix B.2.

4.2 AMR 3.0 results

We compare the performance of SAFT on AMR 3.0 against state-of-the-art AMR-to-text generation baselines: SPRING (Bevilacqua et al., 2021), StructAdapt (Ribeiro et al., 2021), and BiBL (Cheng et al., 2022). All models use the same training data (AMR 3.0), unless noted otherwise, and are evaluated on the identical held-out test split, ensuring a fair comparison. BiBL and SPRING report additional results with extra heuristically labeled data for training; these are grayed out in Table 1. In particular, SPRING and BiBL employ linearization strategies that closely mirror our preprocessing, including comparable traversal orders and handling of edge labels, which further supports the validity of our comparative evaluation.

The results in Table 1 present a comparative evaluation of prior approaches[5] (Bevilacqua et al., 2021; Cheng et al., 2022; Ribeiro et al., 2021), alongside our fine-tuned LLMs, both with and without the proposed graph positional encoding module (SAFT). For baseline models, we include results for versions trained with and without extra heuristically labeled data where available. Notably, we find that fine-tuning several LLMs using LoRA (Hu et al., 2022) already yields improvements over earlier models, including those that use extra training data. Our proposed approach, SAFT, further boosts performance, highlighting the benefit of incorporating structural information in the form of graph positional encodings during LLM fine-tuning. As shown in Table 1, SAFT consistently outperforms both prior baselines and standard fine-tuning (FT), with aggregate BLEU gains of +0.8 over the FT variant across the different models. However, such aggregate scores can obscure important variation

[5]Reported scores are taken directly from the original publications and not reproduced.



(a) Absolute improvement ($\Delta_{\text{BLEU}}(z)$). (b) Relative improvement ($\Delta^1_{\text{BLEU}}(z)$).

Figure 4: BLEU score improvements of structurally-aware fine-tuned (SAFT) models over conventionally fine-tuned (FT) counterparts, on AMR instances of depth $\delta(\mathcal{A}) \geq z$. (a) **Absolute improvement** (Δ_{BLEU}): differences in BLEU score between SAFT and FT models across graph depths and model families. (b) **Relative improvement** (Δ^1_{BLEU}): differences in BLEU scores between SAFT and FT models across graph depths and model families normalized by performance at depth-1 graphs. Both plots reveal a clear and increasing advantage of SAFT as structural complexity grows, demonstrating its effectiveness in leveraging graph topology for improved generation. All lines are second-degree polynomial fits.

across inputs of different structural complexity. To more accurately characterize when and how our method helps, we turn to a stratified analysis in the following sections.

4.3 Complexity-stratified results

To evaluate performance variation with input complexity, we stratify the evaluation by AMR graph depth $\delta(\mathcal{A})$, measured on the original AMR $\mathcal{A}$ prior to preprocessing, by grouping examples where $\delta(\mathcal{A}) \geq z$ for varying thresholds z. We then calculate the BLEU score on these stratified subsets to quantify how our structure-aware fine-tuning approach improves performance at increasing $\delta(\mathcal{A})$ with respect to conventional fine-tuning. We define the following metric:

$$\Delta_{\text{BLEU}}(z) = \text{BLEU}^z_{\text{SAFT}} - \text{BLEU}^z_{\text{FT}},$$

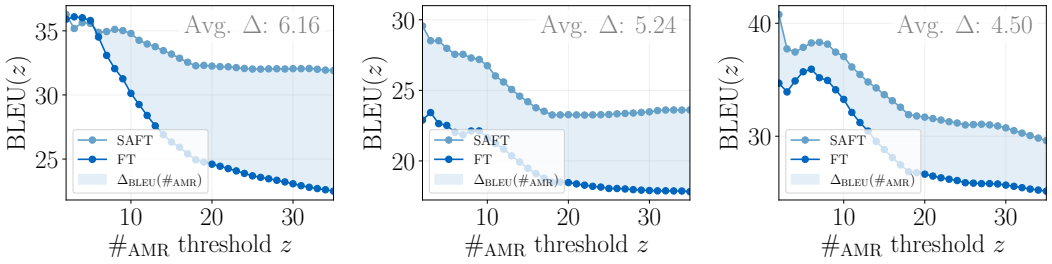
where $\text{BLEU}^z_{\text{SAFT}}$ and $\text{BLEU}^z_{\text{FT}}$ represent the BLEU scores achieved by SAFT and conventional fine-tuning (FT), respectively, on instances with AMR depth $\delta(\mathcal{A}) \geq z$. Fig. 4a shows the extent to which SAFT improves the performance of LLMs at increasing levels of semantic complexity. On semantically complex AMRs (depth $\delta(\mathcal{A}) \geq 8$), SAFT surpasses FT by +1.1 to +4.4 BLEU (Fig. 4a).

Table 1: **SAFT achieves state-of-the-art AMR-to-text generation.** We report BLEU/CHRF++ on AMR 3.0, comparing: (1) prior work, and (2) our LLMs (FT vs. SAFT). All models use identical training data unless marked '+ '. Best results per metric (excluding those using additional data) are highlighted in bold.

Model	Variant	BLEU ↑	CHRF++ ↑
Previous Work			
BiBL	w/o Extra data	47.4	74.5
	+ Extra data	50.7	76.7
SPRING	w/o Extra data	44.9	72.9
	+ Extra data	46.5	73.9
StructAdapt	w/o Extra data	48.0	73.2
Our Finetuned LLMs: trained without extra data			
LLaMA 3.2 (3B)	FT	53.5	75.5
	SAFT	**54.2** (+1.3%)	**76.0** (+0.7%)
LLaMA 3.2 (1B)	FT	45.5	70.9
	SAFT	47.8 **(+5.1%)**	71.9 (+1.4%)
Qwen 2.5 (3B)	FT	51.6	72.1
	SAFT	51.9 (+0.6%)	74.8 **(+3.7%)**
Qwen 2.5 (1.5B)	FT	50.5	73.7
	SAFT	51.7 (+2.4%)	74.5 (+1.1%)
Qwen 2.5 (0.5B)	FT	42.7	69.0
	SAFT	42.9 (+0.5%)	69.3 (+0.4%)
Gemma (2B)	FT	52.9	73.5
	SAFT	52.9 (+0.1%)*	73.6 (+0.1%)

*Not rounded scores: 52.87 (FT) vs. 52.91 (SAFT).

The divergence of the curves at greater depths and the consistent upward trend across all models highlight the increasing importance of modelling structure explicitly as semantic complexity grows. While Gemma 2B



(a) Qwen 2.5 3B (b) LLaMA 3.2 1B (c) LLaMA 3.2 3B

Figure 5: **SAFT demonstrates increasing gains over standard fine-tuning (FT) as document complexity increases.** Performance on the DocAMR test set: Each plot shows the BLEU score improvement of SAFT over FT models, evaluated cumulatively on document-level AMRs with $\#_{\text{AMR}} \leq z$, where $\#_{\text{AMR}}$ denotes the number of AMR graphs contained in a document. This bottom-up stratified evaluation reveals how SAFT performs on increasingly complex document structures. On average across document sizes, SAFT outperforms FT models by +6.16, +5.24, and +4.50 BLEU for Qwen 2.5 3B, LLaMA 3.2 1B, and LLaMA 3.2 3B, respectively.

showed little overall improvement across the full dataset (see Table 1), the benefit of SAFT becomes more pronounced on examples with greater semantic complexity.

To further assess how the benefit varies with respect to depth-one AMRs (i.e., $\delta(\mathcal{A}) = 1$), we define a second-order delta which measures the change in improvement relative to these simple structures:

$$\Delta^1_{\text{BLEU}}(z) = \Delta_{\text{BLEU}}(z) - \Delta_{\text{BLEU}}(1).$$

Fig. 4b further validates the finding that SAFT delivers increasing gains on more complex AMRs. Specifically, it shows the relative improvement of each model compared to its own performance on depth-one AMRs. The positive upward trends across all models indicate that the advantage of incorporating graph structure grows with semantic complexity. This consistent behavior across model families and sizes reinforces the scalability and applicability of our method across different architectures and parameter scales.

4.4 DocAMR results

To understand even further how SAFT impacts performance on highly complex AMRs, we evaluate both SAFT and standard fine-tuning (FT) on DocAMR, a supplementary subset of AMR 3.0 (Knight et al., 2020) whose test set consists of sentence-level AMRs from the standard AMR test split, merged into documents with annotated coreference edges spanning sentences. Notably, we evaluate models that were conventionally fine-tuned (FT) and structurally-aware fine-tuned (SAFT) on sentence-level AMRs (AMR 3.0), testing their performance in a zero-shot setting on document-level AMRs (DocAMR). This setup allows us to assess the models' ability to generalize to cross-sentence semantics without explicit document-level supervision.

As shown in Fig. 5, SAFT consistently and significantly improves performance across all levels of complexity which is measured by $\#_{\text{AMR}}$, the number of AMR graphs contained within a document-level AMR, indicating the overall size and structural density of the input. While the downward trend shows that all models struggle with deep topologies, the consistent, and occasionally increasing, improvement shows that using SAFT improves performance on more structurally-dense inputs. The performance gap between SAFT and conventional fine-tuning (FT) widens with increasing AMR complexity, suggesting that structural inductive bias becomes increasingly crucial in complex document-level generation. This reinforces our central claim that structure-aware fine-tuning is especially beneficial when models must reason over longer contexts and inter-sentential relations. We excluded the Gemma model from this experiment due to its limited context window (4096 tokens), which could not accommodate DocAMR inputs.

Results summary. Conventional fine-tuning (FT) of LLMs already surpasses prior non-LLM baselines on AMR-to-text generation. Our structure-aware fine-tuning (SAFT) method, further improves performance across model families and scales. The improvements are especially consistent

on semantically complex and document-level inputs, confirming that graph-based positional encodings enhance the model's ability to reason over AMR structure.

5 Related Work

To our knowledge, this is the first work to inject graph positional encodings into LLMs for structure-aware fine-tuning without modifying model architecture. Prior work on graph-to-text (specifically AMR-to-text) generation falls into two main categories: **linearization-based** and **adapter-based**. We refer to Appendix F for further details.

Linearization-based. These approaches serialize graphs into sequences for use with seq2seq models like BART or T5. SPRING (Bevilacqua et al., 2021), AMR-BART (Bai et al., 2022), and BiBL (Cheng et al., 2022) differ in traversal strategies and auxiliary tasks. LLMs have also been applied via fine-tuning (Raut et al., 2025; Mager et al., 2020) or prompting (Yao et al., 2024a; Jin et al., 2024b) over linearized AMRs. Related work extends to molecular graphs (Zheng et al., 2024), tables (Fang et al., 2024), and 3D meshes (Wang et al., 2024).

Adapter-based. These methods inject structure via graph-native components like GCN- or GNN-based adapters that embed relational information. StructAdapt (Ribeiro et al., 2021) introduces graph-aware adapters within pretrained transformers to enable reasoning over graph topology. Others modify attention mechanisms to model AMR structure (Zhu et al., 2019). Some methods train graph2seq models (Song et al., 2018; Wang et al., 2020) that natively process graphs without pretrained seq2seq backbones.

6 Conclusion

We introduce SAFT, a structure-aware fine-tuning strategy that injects relational inductive bias into LLMs using graph positional encodings derived from AMR structures. Applied to AMR-to-text generation—a challenging task requiring deep semantic understanding—our approach consistently improves generation quality over conventional fine-tuning and non-LLM-based baselines. We find that performance gains are most pronounced as AMR complexity increases, indicating that structural guidance is particularly valuable for modeling long-range dependencies and rich graph semantics. These results hold across both sentence-level (AMR 3.0) and document-level (DocAMR) benchmarks. Our findings demonstrate that integrating structural signals into LLMs can enhance their reasoning over graph-structured inputs, and we believe this opens the door to broader applications of graph-aware fine-tuning across graph-to-text and other graph-centric tasks.

References

Miltiadis Allamanis, Marc Brockschmidt, and Mahmoud Khademi. Learning to represent programs with graphs. *arXiv preprint arXiv:1711.00740*, 2017.

Jinze Bai, Shuai Bai, Yunfei Chu, Zeyu Cui, Kai Dang, Xiaodong Deng, Yang Fan, Wenbin Ge, Yu Han, Fei Huang, Binyuan Hui, Luo Ji, Mei Li, Junyang Lin, Runji Lin, Dayiheng Liu, Gao Liu, Chengqiang Lu, Keming Lu, Jianxin Ma, Rui Men, Xingzhang Ren, Xuancheng Ren, Chuanqi Tan, Sinan Tan, Jianhong Tu, Peng Wang, Shijie Wang, Wei Wang, Shengguang Wu, Benfeng Xu, Jin Xu, An Yang, Hao Yang, Jian Yang, Shusheng Yang, Yang Yao, Bowen Yu, Hongyi Yuan, Zheng Yuan, Jianwei Zhang, Xingxuan Zhang, Yichang Zhang, Zhenru Zhang, Chang Zhou, Jingren Zhou, Xiaohuan Zhou, and Tianhang Zhu. Qwen technical report. *arXiv preprint arXiv:2309.16609*, 2023.

Xuefeng Bai, Yulong Chen, and Yue Zhang. Graph Pre-training for AMR Parsing and Generation. In Smaranda Muresan, Preslav Nakov, and Aline Villavicencio, editors, *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 6001–6015, Dublin, Ireland, May 2022. Association for Computational Linguistics. doi: 10.18653/v1/2022.acl-long.415.

Laura Banarescu, Claire Bonial, Shu Cai, Madalina Georgescu, Kira Griffitt, Ulf Hermjakob, Kevin Knight, Philipp Koehn, Martha Palmer, and Nathan Schneider. Abstract Meaning Representation

for Sembanking. In Antonio Pareja-Lora, Maria Liakata, and Stefanie Dipper, editors, *Proceedings of the 7th Linguistic Annotation Workshop and Interoperability with Discourse*, pages 178–186, Sofia, Bulgaria, August 2013. Association for Computational Linguistics.

John Bateman. Upper modeling: Organizing knowledge for natural language processing. In *Proceedings of the Fifth International Workshop on Natural Language Generation*, 1990.

Mikhail Belkin and Partha Niyogi. Laplacian Eigenmaps for Dimensionality Reduction and Data Representation. *Neural Computation*, 15(6):1373–1396, 2003. doi: 10.1162/089976603321780317.

Michele Bevilacqua, Rexhina Blloshmi, and Roberto Navigli. One SPRING to Rule Them Both: Symmetric AMR Semantic Parsing and Generation without a Complex Pipeline. *Proceedings of the AAAI Conference on Artificial Intelligence*, 35(14):12564–12573, May 2021. ISSN 2159-5399. doi: 10.1609/aaai.v35i14.17489.

Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D 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 Ziegler, Jeffrey Wu, Clemens Winter, Chris Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. Language Models are Few-Shot Learners. In H. Larochelle, M. Ranzato, R. Hadsell, M.F. Balcan, and H. Lin, editors, *Advances in Neural Information Processing Systems*, volume 33, pages 1877–1901. Curran Associates, Inc., 2020.

Ziming Cheng, Zuchao Li, and Hai Zhao. BiBL: AMR Parsing and Generation with Bidirectional Bayesian Learning. In Nicoletta Calzolari, Chu-Ren Huang, Hansaem Kim, James Pustejovsky, Leo Wanner, Key-Sun Choi, Pum-Mo Ryu, Hsin-Hsi Chen, Lucia Donatelli, Heng Ji, Sadao Kurohashi, Patrizia Paggio, Nianwen Xue, Seokhwan Kim, Younggyun Hahm, Zhong He, Tony Kyungil Lee, Enrico Santus, Francis Bond, and Seung-Hoon Na, editors, *Proceedings of the 29th International Conference on Computational Linguistics*, pages 5461–5475, Gyeongju, Republic of Korea, oct 2022. International Committee on Computational Linguistics.

Tianyu Cui, Xinjie Lin, Sijia Li, Miao Chen, Qilei Yin, Qi Li, and Ke Xu. TrafficLLM: Enhancing Large Language Models for Network Traffic Analysis with Generic Traffic Representation. *arXiv preprint arXiv:2504.04222*, 2025.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. In *Proceedings of the 2019 conference of the North American chapter of the association for computational linguistics: human language technologies, volume 1 (long and short papers)*, pages 4171–4186, 2019.

Vijay Prakash Dwivedi and Xavier Bresson. A Generalization of Transformer Networks to Graphs. *AAAI Workshop on Deep Learning on Graphs: Methods and Applications*, 2021.

Xi Fang, Weijie Xu, Fiona Anting Tan, Jiani Zhang, Ziqing Hu, Yanjun Qi, Scott Nickleach, Diego Socolinsky, Srinivasan Sengamedu, and Christos Faloutsos. Large Language Models(LLMs) on Tabular Data: Prediction, Generation, and Understanding – A Survey. *arXiv preprint arXiv:2402.17944*, 2024.

Bahare Fatemi, Jonathan Halcrow, and Bryan Perozzi. Talk like a Graph: Encoding Graphs for Large Language Models. In *The Twelfth International Conference on Learning Representations*, 2024.

Satoshi Furutani, Toshiki Shibahara, Mitsuaki Akiyama, Kunio Hato, and Masaki Aida. Graph Signal Processing for Directed Graphs Based on the Hermitian Laplacian. In Ulf Brefeld, Elisa Fromont, Andreas Hotho, Arno Knobbe, Marloes Maathuis, and Céline Robardet, editors, *Machine Learning and Knowledge Discovery in Databases*, pages 447–463, Cham, 2020. Springer International Publishing. ISBN 978-3-030-46150-8.

Simon Geisler, Yujia Li, Daniel J Mankowitz, Ali Taylan Cemgil, Stephan Günnemann, and Cosmin Paduraru. Transformers Meet Directed Graphs. In Andreas Krause, Emma Brunskill, Kyunghyun Cho, Barbara Engelhardt, Sivan Sabato, and Jonathan Scarlett, editors, *Proceedings of the 40th International Conference on Machine Learning*, volume 202, pages 11144–11172. PMLR, 23–29 Jul 2023.

Michael Wayne Goodman. Penman: An open-source library and tool for AMR graphs. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics: System Demonstrations*, pages 312–319, 2020.

Dan Hendrycks and Kevin Gimpel. Gaussian Error Linear Units (GELUs). *arXiv preprint arXiv:1606.08415*, 2016.

Edward J Hu, yelong shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen. LoRA: Low-Rank Adaptation of Large Language Models. In *International Conference on Learning Representations*, 2022.

Jinhao Jiang, Kun Zhou, Zican Dong, Keming Ye, Xin Zhao, and Ji-Rong Wen. StructGPT: A General Framework for Large Language Model to Reason over Structured Data. In Houda Bouamor, Juan Pino, and Kalika Bali, editors, *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pages 9237–9251, Singapore, Dec 2023. Association for Computational Linguistics. doi: 10.18653/v1/2023.emnlp-main.574.

Bowen Jin, Gang Liu, Chi Han, Meng Jiang, Heng Ji, and Jiawei Han. Large Language Models on Graphs: A Comprehensive Survey. *IEEE Transactions on Knowledge and Data Engineering*, 36 (12):8622–8642, 2024a. doi: 10.1109/TKDE.2024.3469578.

Zhijing Jin, Yuen Chen, Fernando Gonzalez Adauto, Jiarui Liu, Jiayi Zhang, Julian Michael, Bernhard Schölkopf, and Mona Diab. Analyzing the Role of Semantic Representations in the Era of Large Language Models. In *Proceedings of the 2024 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (Volume 1: Long Papers)*, pages 3781–3798, 2024b.

Robert T Kasper. 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.

Kevin Knight, Bianca Badarau, Laura Baranescu, Claire Bonial, Madalina Bardocz, Kira Griffitt, Ulf Hermjakob, Daniel Marcu, Martha Palmer, Tim O'Gorman, and Nathan Schneider. Abstract Meaning Representation (AMR) Annotation Release 2.0. https://catalog.ldc.upenn.edu/LDC2017T10, 2017. LDC2017T10, Web Download. Philadelphia: Linguistic Data Consortium.

Kevin Knight, Bianca Badarau, Laura Baranescu, Claire Bonial, Madalina Bardocz, Kira Griffitt, Ulf Hermjakob, Daniel Marcu, Martha Palmer, Tim O'Gorman, and Nathan Schneider. Abstract Meaning Representation (AMR) Annotation Release 3.0. https://catalog.ldc.upenn.edu/LDC2020T02, 2020. LDC2020T02, Web Download. Philadelphia: Linguistic Data Consortium.

Ioannis Konstas, Srinivasan Iyer, Mark Yatskar, Yejin Choi, and Luke Zettlemoyer. Neural AMR: Sequence-to-Sequence Models for Parsing and Generation. In Regina Barzilay and Min-Yen Kan, editors, *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 146–157, Vancouver, Canada, Jul 2017. Association for Computational Linguistics. doi: 10.18653/v1/P17-1014.

Irene Langkilde and Kevin Knight. Generation that exploits corpus-based statistical knowledge. In *COLING 1998 Volume 1: The 17th International Conference on Computational Linguistics*, 1998.

Manuel Mager, Ramón Fernandez Astudillo, Tahira Naseem, Md Arafat Sultan, Young-Suk Lee, Radu Florian, and Salim Roukos. GPT-too: A Language-Model-First Approach for AMR-to-Text Generation. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 1846–1852, 2020.

Behrooz Mansouri. Survey of Abstract Meaning Representation: Then, Now, Future. *arXiv preprint arXiv:505.03229*, 2025.

Kishore Papineni, Salim Roukos, Todd Ward, and Wei-Jing Zhu. Bleu: a Method for Automatic Evaluation of Machine Translation. In *Proceedings of the 40th annual meeting of the Association for Computational Linguistics*, pages 311–318, 2002.

Maja Popović. chrF: character n-gram F-score for automatic MT evaluation. In *Proceedings of the Tenth Workshop on Statistical Machine Translation*, pages 392–395, 2015.

Ankush Raut, Xiaofeng Zhu, and Maria Leonor Pacheco. Can LLMs Interpret and Leverage Structured Linguistic Representations? A Case Study with AMRs. *arXiv preprint arXiv:2504.04745*, 2025.

Leonardo F. R. Ribeiro, Yue Zhang, and Iryna Gurevych. Structural Adapters in Pretrained Language Models for AMR-to-Text Generation. In Marie-Francine Moens, Xuanjing Huang, Lucia Specia, and Scott Wen-tau Yih, editors, *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*, pages 4269–4282, Online and Punta Cana, Dominican Republic, November 2021. Association for Computational Linguistics. doi: 10.18653/v1/2021.emnlp-main.351.

Linfeng Song, Yue Zhang, Zhiguo Wang, and Daniel Gildea. A Graph-to-Sequence Model for AMR-to-Text Generation. In Iryna Gurevych and Yusuke Miyao, editors, *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1616–1626, Melbourne, Australia, jul 2018. Association for Computational Linguistics. doi: 10.18653/v1/P18-1150.

Jiabin Tang, Yuhao Yang, Wei Wei, Lei Shi, Lixin Su, Suqi Cheng, Dawei Yin, and Chao Huang. Graphgpt: Graph instruction tuning for large language models. In *Proceedings of the 47th International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 491–500, 2024.

Gemma Team, Thomas Mesnard, Cassidy Hardin, Robert Dadashi, Surya Bhupatiraju, Shreya Pathak, Laurent Sifre, Morgane Rivière, Mihir Sanjay Kale, Juliette Love, Pouya Tafti, Léonard Hussenot, Pier Giuseppe Sessa, Aakanksha Chowdhery, Adam Roberts, Aditya Barua, Alex Botev, Alex Castro-Ros, Ambrose Slone, Amélie Héliou, Andrea Tacchetti, Anna Bulanova, Antonia Paterson, Beth Tsai, Bobak Shahriari, Charline Le Lan, Christopher A. Choquette-Choo, Clément Crepy, Daniel Cer, Daphne Ippolito, David Reid, Elena Buchatskaya, Eric Ni, Eric Noland, Geng Yan, George Tucker, George-Christian Muraru, Grigory Rozhdestvenskiy, Henryk Michalewski, Ian Tenney, Ivan Grishchenko, Jacob Austin, James Keeling, Jane Labanowski, Jean-Baptiste Lespiau, Jeff Stanway, Jenny Brennan, Jeremy Chen, Johan Ferret, Justin Chiu, Justin Mao-Jones, Katherine Lee, Kathy Yu, Katie Millican, Lars Lowe Sjoesund, Lisa Lee, Lucas Dixon, Machel Reid, Maciej Mikuła, Mateo Wirth, Michael Sharman, Nikolai Chinaev, Nithum Thain, Olivier Bachem, Oscar Chang, Oscar Wahltinez, Paige Bailey, Paul Michel, Petko Yotov, Rahma Chaabouni, Ramona Comanescu, Reena Jana, Rohan Anil, Ross McIlroy, Ruibo Liu, Ryan Mullins, Samuel L Smith, Sebastian Borgeaud, Sertan Girgin, Sholto Douglas, Shree Pandya, Siamak Shakeri, Soham De, Ted Klimenko, Tom Hennigan, Vlad Feinberg, Wojciech Stokowiec, Yu hui Chen, Zafarali Ahmed, Zhitao Gong, Tris Warkentin, Ludovic Peran, Minh Giang, Clément Farabet, Oriol Vinyals, Jeff Dean, Koray Kavukcuoglu, Demis Hassabis, Zoubin Ghahramani, Douglas Eck, Joelle Barral, Fernando Pereira, Eli Collins, Armand Joulin, Noah Fiedel, Evan Senter, Alek Andreev, and Kathleen Kenealy. Gemma: Open Models Based on Gemini Research and Technology. *arXiv preprint arXiv:2403.08295*, 2024.

Yijun Tian, Huan Song, Zichen Wang, Haozhu Wang, Ziqing Hu, Fang Wang, Nitesh V Chawla, and Panpan Xu. Graph neural prompting with large language models. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 38, pages 19080–19088, 2024.

Hugo Touvron, Thibaut Lavril, Gautier Izacard, Xavier Martinet, Marie-Anne Lachaux, Timothée Lacroix, Baptiste Rozière, Naman Goyal, Eric Hambro, Faisal Azhar, et al. LLama: Open and efficient foundation language models. *arXiv preprint arXiv:2302.13971*, 2023.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. In I. Guyon, U. Von Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc., 2017.

Tianming Wang, Xiaojun Wan, and Hanqi Jin. AMR-To-Text Generation with Graph Transformer. *Transactions of the Association for Computational Linguistics*, 8:19–33, 2020.

Zhengyi Wang, Jonathan Lorraine, Yikai Wang, Hang Su, Jun Zhu, Sanja Fidler, and Xiaohui Zeng. LLaMA-Mesh: Unifying 3D Mesh Generation with Language Models. *arXiv preprint arXiv:2411.09595*, 2024.

Xu Yang, Kaihua Tang, Hanwang Zhang, and Jianfei Cai. Auto-encoding Scene Graphs for Image Captioning. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 10685–10694, 2019.

Peiran Yao, Kostyantyn Guzhva, and Denilson Barbosa. Semantic Graphs for Syntactic Simplification: A Revisit from the Age of LLM. *Proceedings of TextGraphs-17: Graph-based Methods for Natural Language Processing*, page 105, 2024a.

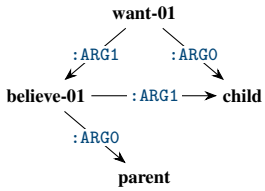
Yang Yao, Xin Wang, Zeyang Zhang, Yijian Qin, Ziwei Zhang, Xu Chu, Yuekui Yang, Wenwu Zhu, and Hong Mei. Exploring the potential of large language models in graph generation. *arXiv preprint arXiv:2403.14358*, 2024b.

Xikun Zhang, Antoine Bosselut, Michihiro Yasunaga, Hongyu Ren, Percy Liang, Christopher D Manning, and Jure Leskovec. GreaseLM: Graph REASoning Enhanced Language Models. In *International Conference on Learning Representations*, 2022.

Yizhen Zheng, Huan Yee Koh, Maddie Yang, Li Li, Lauren T. May, Geoffrey I. Webb, Shirui Pan, and George Church. Large Language Models in Drug Discovery and Development: From Disease Mechanisms to Clinical Trials. *arXiv preprint arXiv:2409.04481*, 2024.

Jie Zhu, Junhui Li, Muhua Zhu, Longhua Qian, Min Zhang, and Guodong Zhou. Modeling Graph Structure in Transformer for Better AMR-to-Text Generation. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, pages 5459–5468, 2019.

A Abstract Meaning Representation



(a) Graph representation

```
(w / want-01
  :ARG0 (c / child)
  :ARG1 (b / believe-01
    :ARG0 (p / parent)
    :ARG1 c))
```

(b) Penman notation

```
<P0> want-01 :ARG0 <P1> child
:ARG1 <P2> believe-01 <stop>
<P2> :ARG0 <P3> parent :ARG1
<P1> <stop>
```

(c) BFS linearization

Figure 6: Three aligned representations of the sentence *"The child wants the parent to believe them."*: (a) a graph-based AMR structure, (b) its corresponding Penman notation, and (c) a BFS linearization used for sequence-based processing.

A.1 Reference sentence

We report here the representations of the sentence used in the main body (Section 2.2) as reference:

> *The child wants the parent to believe them.*

Fig. 6a is the graph representation of the sentence, while Figs. 6b and 6c are the Penman and BFS linearizations, respectively.

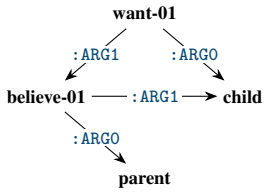
B Implementation details

B.1 Semantically-preserving AMR transformation

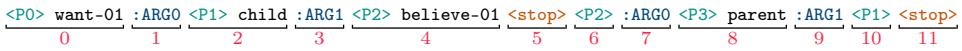
We report detailed information of our proposed semantically-preserving transformation $\tau : \Sigma^* \times (\Sigma \rightarrowtail \mathcal{V}_\mathcal{G}) \rightarrow \mathbb{G}$ of AMR graph (Section 3.1).

Given a linearization (label sequence) $\mathcal{L}_\mathcal{A}$ and alignment $\sigma_\mathcal{A}$—as discussed in Section 3.1 and shown in Fig. 7b—the transformation τ constructs the SPG $\mathcal{G}_\mathcal{A} = \tau(\mathcal{L}_\mathcal{A}, \sigma_\mathcal{A})$ through the following steps:

1. **Substructure Construction** (ToSubgraph, Fig. 7c): $\mathcal{L}_\mathcal{A}$ is segmented at each `<stop>` token. Each segment defines a local subgraph rooted at a head concept and includes its outgoing role-labeled edges (e.g., `:ARG0`) and target nodes.
$$\{\bar{\mathcal{A}}_i\}_i = \textsc{ToSubgraph}(\mathcal{L}_\mathcal{A}).$$
2. **Edge-to-Node Conversion** (RoleExpand, Fig. 7d): Each labeled edge $(u \xrightarrow{r} v)$ is expanded into a role node r, creating two unlabeled edges: $(u \rightarrow r)$ and $(r \rightarrow v)$. This yields a directed graph with no edge labels.
$$\bar{\mathcal{G}}_i = \textsc{RoleExpand}(\bar{\mathcal{A}}_i).$$
3. **Stop Node Re-insertion** (AddStopNodes, Fig. 7d): The `<stop>` labels are inserted in each subgraph as a special terminal node. These nodes mark the end of node expansions and, alongside $\sigma_\mathcal{A}$, support alignment between graph nodes and tokens in $\mathcal{L}_\mathcal{A}$.
$$\hat{\mathcal{G}}_i = \textsc{AddStopNodes}(\bar{\mathcal{G}}_i).$$
4. **Node Ordering Assignment** ($\sigma_\mathcal{A}^{-1}$, Fig. 7d): Assign to each node an index inherited from the BFS order to preserve alignment between token positions in $\mathcal{L}_\mathcal{A}$ and graph nodes in $\mathcal{G}$.
$$i_v = \sigma_\mathcal{A}^{-1}(v), \quad \forall v \in \hat{\mathcal{G}}_i.$$
5. **Pointer-Based Merging** (Merge, Fig. 7e): For each pointer index j (e.g., `<P2>`), identify co-referring nodes $\mathcal{U}_j = \{u_1, \ldots, u_k\}$ such that all u_i share pointer j. Then:
   (a) Merge *incoming* edges: $\mathcal{E}^{\text{in}}_{\mathcal{U}_j} = \bigcup_{i=1}^{k} \mathcal{E}^{\text{in}}(u_i)$, with $\mathcal{E}^{\text{in}}(u_i) = \{(v, u_i) : (v, u_i) \in \mathcal{E}_\mathcal{A}\}$.



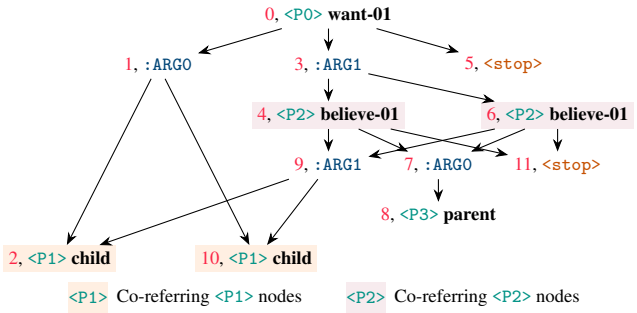
(a) Input AMR graph $\mathcal{A}$



(b) $\mathcal{L}_{\mathcal{A}} = (\ell_1, \ldots, \ell_L) = \text{BFS}(\mathcal{A})$, $\sigma_{\mathcal{A}} : \mathbb{Z} \rightarrowtail \mathcal{V}_{\mathcal{G}}$

(c) ToSubgraph

(d) RoleExpand, AddStopNodes, and $\sigma_{\mathcal{A}}^{-1}$



(e) Merge

Figure 7: Transformation steps from an AMR graph $\mathcal{A}$ to the Role-Expanded Graph $\mathcal{G}$.

(b) Merge *outgoing* edges: $\mathcal{E}^{\text{out}}_{\mathcal{U}_j} = \bigcup_{i=1}^{k} \mathcal{E}^{\text{out}}(u_i)$, with $\mathcal{E}^{\text{out}}(u_i) = \{(u_i, v) : (u_i, v) \in \mathcal{E}_{\mathcal{A}}\}$

(c) Update the connectivity for each $u_i \in \mathcal{U}_j$: $\mathcal{E}^{\text{in}}(u_i) := \mathcal{E}^{\text{in}}_{\mathcal{U}_j}$, $\quad \mathcal{E}^{\text{out}}(u_i) := \mathcal{E}^{\text{out}}_{\mathcal{U}_j}$

$$\mathcal{G}_{\mathcal{A}} = \text{Merge}(\{\bar{\mathcal{G}}_i\}_i).$$

B.2 Training details

Training setup. To fine-tune pretrained LLMs for this task, we adapt the open-source `LitGPT`[6] codebase as a base framework and add our modifications to it. Our modifications include:

- Integration of graph-based positional encodings;
- Support for token-node alignment during positional embedding construction through special nodewise tokenization;

[6] https://github.com/Lightning-AI/litgpt

- Initializing the function f_θ used for projecting the graph positional encodings;
- Adding custom prompts for our task.

Tokenizer. We experimented with adding AMR role labels (e.g., `:ARG0`, `:ARG1`, `:mod`) to the tokenizer and extending the model's vocabulary accordingly. However, we found that the default tokenizer yielded more stable performance, suggesting that extending the vocabulary with role labels did not offer additional benefits. Therefore, we retain the original tokenizer throughout all experiments.

Hardware setup. All models were trained on a single GPU node with 64 GB of RAM. Models with 2 billion parameters or more were trained on an NVIDIA H100 GPU, while smaller models (< 2B parameters) were trained on an A100 GPU.

B.2.1 Hyperparameters

The hyperparameter choice for each model can be found in Table 2 and Table 3.

Table 2: Hyperparameter configurations for each SAFT models

Category	Hyperparameter	LLaMA 1B	LLaMA 3.2B	Qwen 0.5B	Qwen 1.5B	Qwen 3B	Gemma 2B
LoRA	Rank (r)	32	32	32	32	16	32
	Scaling factor (α)	32	64	32	64	16	32
	Dropout	0.05	0.05	0.05	0.05	0.05	0.05
	Head enabled	True	True	True	True	True	True
Training	Epochs	6	5	6	5	6	5
	Warmup steps	100	100	100	100	100	100
	Effective batch size	256	256	256	256	256	256
Custom	# Eigenvectors (k)	30	30	30	30	30	25
	MLP LR Multiplier (μ)	0.8	0.8	0.9	0.8	0.8	0.5
	Magnetic param (q)	0.25	0.25	0.25	0.25	0.25	0.25
	Sinusoidal base (q_{sin})	1000	1000	1000	1000	1000	1000
	Sinusoidal dim (d)	8	8	8	8	8	8

Table 3: Hyperparameter configurations for each conventionally fine-tuned model.

Category	Hyperparameter	LLaMA 1B	LLaMA 3.2B	Qwen 0.5B	Qwen 1.5B	Qwen 3B	Gemma 2B
LoRA	Rank (r)	16	8	16	32	16	32
	Scaling factor (α)	16	8	16	32	16	32
	Dropout	0.05	0.05	0.05	0.05	0.05	0.05
	Head enabled	True	True	True	True	True	True
Training	Epochs	5	5	8	6	8	5
	Warmup steps	100	100	100	100	100	100
	Effective batch size	256	256	256	256	256	256

LoRA Hyperparameters. Given the high computational cost of fine-tuning, we adopted a practical manual hyperparameter search strategy focused on LoRA configurations. We used all LoRA layer types (query, key, value, projection, and head) by default, with a rank (r) and scaling factor (α) chosen from $\{4, 8, 16, 32\}$. Dropout rates were selected from $\{0.05, 0.1, 0.15\}$. In cases where overfitting was observed, we first adjusted the dropout rate to improve generalization. If overfitting persisted, we disabled the LoRA head component, which we found to be the least critical for performance in preliminary runs. This strategy allowed us to balance empirical effectiveness with computational feasibility.

Epochs and training time. All models were trained for 10 epochs with checkpoints saved at the end of each epoch, and the one with the best validation BLEU was chosen (the number of epochs reported is the one with the best BLEU). Training time varied from 9 hours for the smallest models to 16 hours for the larger ones.

Leaning rate. We use a learning rate schedule with linear warmup for the first 100 optimizer steps, followed by cosine annealing until the end of training.

Custom hyperparameters. There are five hyperparameters that are specific to our approach:

- **Number of eigenvectors** (k): the number of eigenvectors used as positional encodings; we select the k eigenvectors corresponding to the smallest k eigenvalues. We found that the performance is most stable in the range of 20 to 40 eigenvectors and therefore we chose from {20,25,30,35,40}.
- **MLP learning rate multiplier** (μ): to improve training stability, we scale the learning rate of the MLP projecting positional encodings by a constant factor μ, applied on top of the scheduled learning rate; that is, $\mathrm{LR}_{f_\theta}(t) = \mu \cdot \mathrm{LR}(t)$, where $\mathrm{LR}(t)$ is the base learning rate at step t.
- **Magnetic parameter** (q): controls the strength of the complex rotation in the magnetic Laplacian, modulating the influence of edge directionality. After experimenting with values between 10^{-3} and 0.5, we found $q = 0.25$ yielded the most stable results and fixed it for most experiments.
- **Sinusoidal PE frequency base** (q_{sin}): the base used in the frequency scaling of sinusoidal positional encodings, analogous to that in Transformer models. Since inter-node sequences are relatively short in our setting, we use $q_{\text{sin}} = 1000$.
- **Sinusoidal PE dimension** (d): defines the number of features in the sinusoidal positional encodings concatenated with the eigenvector-based encodings. We set this to 8.

Models. We used Low-Rank Adaptation (LoRA) (Hu et al., 2022) to fine-tune the following pretrained LLMs: LLaMA 3.2 (3B and 1B) (Touvron et al., 2023), Qwen 2.5 (3B, 1.5B, and 0.5B) (Bai et al., 2023), Gemma 2B (Team et al., 2024). We also attempted to fine-tune Gemma 7B, but encountered frequent out-of-memory (OOM) issues when training on longer AMR sequences, which limited its usability in our experiments. For each model, we compare two variants: one fine-tuned with our positional encodings (PEs) integrated during training, and one without. For both variants, we report results using the best-performing checkpoint found during development. During evaluation, the PEs are activated consistently based on the corresponding training configuration.

Prompting Format. To enable AMR-to-text generation with instruction-tuned large language models, we adopt a structured prompting format implemented via the `AMR2Text` prompt style. Each prompt consists of three components:

- A **starting token**, which includes task metadata and generation instructions:

```
<AMR-to-Text>
[Task: AMR-to-Text]
[Instruction] Convert the following AMR into natural language text.
[Input: AMR]
```

- The **linearized AMR graph** $\mathcal{L}_{\mathcal{A}}$, inserted directly after the input header. This is a token sequence derived from the input AMR graph $\mathcal{A}$ (see Section 3.1).
- An **ending token**, marking the beginning of the generation segment:

```
[Output: Text]
```

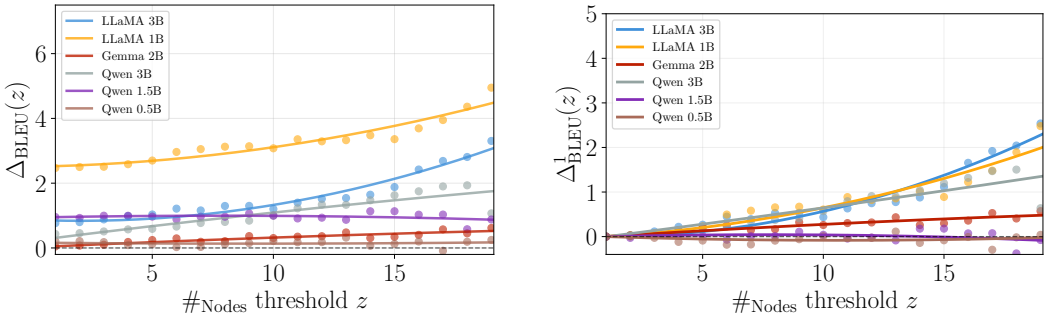
The full prompt passed to the model is thus structured as:

```
<AMR-to-Text>
[Task: AMR-to-Text]
[Instruction] Convert the following AMR into natural language text.
[Input: AMR]
$\mathcal{L}_{\mathcal{A}}$
[Output: Text]
```

C Additional Experiments

C.1 Stratified evaluation over number of nodes

Similarly to the evaluation in (Section 4.3), we perform a stratified analysis based on the number of nodes in the original AMR graph $\mathcal{A}$ to examine how both graph size and structural complexity



(a) Absolute improvement ($\Delta_{\text{BLEU}}(z)$).

(b) Relative improvement ($\Delta^1_{\text{BLEU}}(z)$).

Figure 8: BLEU score improvements of structurally-aware fine-tuned (SAFT) models over conventionally fine-tuned (FT) counterparts, on AMR instances with number of nodes $\#_{\text{Nodes}}(\mathcal{A}) \geq z$. (a) **Absolute improvement** (Δ_{BLEU}): differences in BLEU score between SAFT and FT models across varying number of nodes and model families. (b) **Relative improvement** (Δ^1_{BLEU}): differences in BLEU scores normalized by single-node graph performance. The results show consistent gains, though the magnitude of improvement is less pronounced compared to depth-based stratification (see Fig. 4), indicating that structural complexity plays a more critical role than graph size alone. All lines are second-degree polynomial fits.

influence the performance of SAFT. As shown in (Fig. 8), SAFT exhibits consistent improvements over standard fine-tuning across most model sizes, particularly for larger models. However, the trend is less pronounced than in Fig. 4, where stratification was based on graph depth. This contrast highlights that the gains from SAFT are more strongly associated with structural complexity and long-range dependencies than with graph size alone, suggesting that structural information yields diminishing returns when applied to merely larger—but not necessarily deeper—graphs.

D Limitations

While our approach achieves consistent improvements, particularly on semantically complex graphs, several limitations remain. First, it introduces computational overhead from graph preprocessing and structural encoding, though this can be mitigated through caching. Second, gains are less pronounced on simpler inputs with limited structural information, suggesting the method's inductive bias is not universally beneficial. Third, effectiveness depends on hyperparameter choices such as positional encodings dimensionality, which may require tuning. Finally, extending this method to other graph-structured tasks requires task-specific node-to-token alignments, adding engineering complexity.

E Assets and Licences

E.1 Datasets

We evaluate our approach on the AMR 3.0 dataset (LDC2020T02[7]) (Knight et al., 2020), which consists of approximately 55k training instances, 1.3k for development, and 1.4k for testing. Compared to earlier versions (Knight et al., 2017), AMR 3.0 includes more diverse graph structures and broader linguistic coverage, providing a rigorous benchmark for AMR-to-text generation.

This release is a semantically annotated corpus of over 59k English sentences drawn from a diverse mix of domains, including broadcast conversation, discussion forums, weblogs, newswire, and fiction. Annotations cover PropBank-style frames, non-core semantic roles, coreference, named entities, modality, negation, quantities, and questions. Sentence-level annotations are represented as rooted, directed acyclic graphs designed to abstract away from surface syntax and emphasize predicate-argument structure.

[7] https://catalog.ldc.upenn.edu/LDC2020T02

For a subset of experiments, we also evaluate on the DocAMR dataset (part of AMR 3.0), which extends AMR to the document level by providing inter-sentence coreference and discourse-level annotations. This enables assessment of long-range semantic dependencies and coherence in multi-sentence generation.

We use the dataset as released by the Linguistic Data Consortium (LDC2020T02), without augmenting with any silver data (i.e., data labeled through heuristic or automated methods). AMR 3.0 is distributed under the LDC User Agreement and is not publicly available; access requires an institutional or individual LDC license. For reference, the release was published on January 15, 2020 and includes contributions from DARPA-funded programs (BOLT, DEFT, MRP, LORELEI) and NSF-supported research.

Table 4: Datasets used for AMR-to-text generation.

Dataset	Size (Train/Dev/Test)	Key Features	License
AMR 3.0	55k / 1.3k / 1.4k	Sentence-level graphs, broad linguistic coverage	LDC Non-Member License
DocAMR	Derived from AMR 3.0	Document-level annotations, coreference, discourse	Same as AMR 3.0

E.2 Models

We conduct experiments using a selection of publicly available pretrained language models with open or research-focused licenses. All models are used strictly for academic purposes, in compliance with their respective licenses.

LitGPT. We build on the `LitGPT` framework,[8] an open-source project released under the Apache License 2.0.[9] It provides modular components for efficient fine-tuning, inference, and reproducibility across large-scale models.

Qwen. Qwen models,[10] developed by Alibaba Cloud, are released under the Apache License 2.0. This permissive open-source license permits modification, distribution, and commercial use, provided appropriate attribution is maintained.

Gemma. Gemma,[11] developed by Google DeepMind, is also licensed under the Apache License 2.0. This allows for both academic and commercial applications and emphasizes interoperability with a wide range of open-source software.

LLaMA 2. LLaMA 2 models,[12] released by Meta, are governed by the LLAMA 2 Community License Agreement. The license permits use, modification, and redistribution, but restricts: (i) Commercial use by entities exceeding 700 million monthly active users without explicit permission from Meta; (ii) Use of LLaMA outputs to train competing large language models. Redistributions must include a notice file, and use is subject to Meta's Acceptable Use Policy.[13]

Table 5: Pretrained models and licensing details.

Model	Provider	License	Notes
LitGPT	Lightning AI	Apache 2.0	Permissive, for training and inference
Qwen	Alibaba Cloud	Apache 2.0	Open-source, commercial use permitted
Gemma	Google DeepMind	Apache 2.0	Open-source, commercial use permitted
LLaMA 2	Meta	LLAMA 2 Community License	Requires license for large-scale commercial use

[8] https://github.com/Lightning-AI/litgpt
[9] http://www.apache.org/licenses/LICENSE-2.0
[10] https://github.com/QwenLM/Qwen
[11] https://github.com/google-deepmind/gemma
[12] https://ai.meta.com/resources/models-and-libraries/llama-downloads/
[13] https://llama.com/use-policy

F Additional information on related work

Prior work on AMR-to-text generation—and more broadly, text generation from graph-structured data—largely follows two paradigms: **Linearization-based approaches** and **Adapter-based approaches**. To the best of our knowledge, no prior work has explored graph positional encodings in this setting.

F.1 Linearization-based approaches

These methods convert the input graph into a linear sequence and fine-tune a pre-trained encoder-decoder transformer (e.g., BART, T5) in a standard seq-to-seq setup.

Bevilacqua et al. (2021) introduced a symmetric framework for AMR parsing and generation by fine-tuning BART on linearized AMR graphs using both DFS and BFS traversals (SPRING). AMR-BART (Bai et al., 2022) builds on SPRING by incorporating self-supervised graph denoising tasks during pretraining, which improves robustness to structural noise. BiBL (Cheng et al., 2022) further extends this line of work by jointly modeling AMR-to-text and text-to-AMR transitions through single-stage multitask learning with auxiliary losses. These models share a common foundation: they linearize the AMR graph and fine-tune a standard transformer. This strategy has also been applied to large language models (LLMs) via fine-tuning (Raut et al., 2025; Mager et al., 2020) or prompting (Yao et al., 2024a; Jin et al., 2024b) using the linearized AMR graph as input.

More generally, the practice of aligning LLMs with structured data through linearization has found success across domains such as molecular generation (Zheng et al., 2024), network traffic analysis (Cui et al., 2025), tabular reasoning (Fang et al., 2024), and 3D mesh processing (Wang et al., 2024).

F.2 Adapter-based approaches

In contrast, adapter-based methods directly model the structure of the input graph using graph neural networks (GNNs) or related components, which are then integrated into transformer architectures.

StructAdapt (Ribeiro et al., 2021) introduces graph-aware adapters based on Graph Convolutional Networks (GCNs), enabling the model to reason over AMR topologies during fine-tuning. Other methods take a similar direction by modifying the attention mechanism to incorporate structural biases from the input graph (Zhu et al., 2019). Another line of work avoids transformer pretraining altogether, instead training graph-to-sequence models from scratch that can natively process graph inputs (Song et al., 2018; Wang et al., 2020).

F.3 LLMs for graph-structured data

The rise of large language models (LLMs) (Vaswani et al., 2017; Devlin et al., 2019; Brown et al., 2020; Touvron et al., 2023) has reshaped NLP. Recently, there has been growing interest in extending LLMs to handle graph-structured inputs (Jin et al., 2024a), particularly in domains like molecules, knowledge graphs, and social networks. Existing methods typically fall into one of three strategies: (*i*) flattening graphs into linear sequences (Jiang et al., 2023; Fatemi et al., 2024; Yao et al., 2024b); (*ii*) modifying the LLM architecture to incorporate graph encoders (Zhang et al., 2022); or (*iii*) generating structure-aware token embeddings that align with LLM representations (Tian et al., 2024; Tang et al., 2024). The latter direction shows promise but introduces additional training complexity due to the need for separate graph encoders and alignment mechanisms.