SAFT: STRUCTURE-AWARE FINE-TUNING OF LLMS FOR AMR-TO-TEXT GENERATION

Anonymous authors
Paper under double-blind review

ABSTRACT

Large Language Models (LLMs) are increasingly applied to tasks involving structured inputs such as Abstract Meaning Representations (AMRs). However, common approaches either linearize graphs, discarding crucial structural cues, or rely on specialized architectures that are incompatible with standard pretrained LLMs. We present SAFT, a structure-aware fine-tuning method that augments LLMs with graph-sensitive positional encodings derived from the magnetic Laplacian of AMRs. These encodings are projected into the LLM embedding space, introducing relational inductive bias without modifying the model architecture. Designed to be applicable across tasks involving graph-structured inputs, we demonstrate its effectiveness on AMR-to-text generation, where SAFT establishes a new state of the art on AMR 3.0 with a +3.5 BLEU improvement over prior baselines. Performance gains grow with graph complexity, highlighting the value of structure-aware representations in enhancing LLM performance.

1 INTRODUCTION

Large Language Models (LLMs) have become the dominant paradigm for natural language processing, 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 the scalability and flexibility of LLMs as pretrained, general-purpose sequence models.

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 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 represents 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 semantic 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

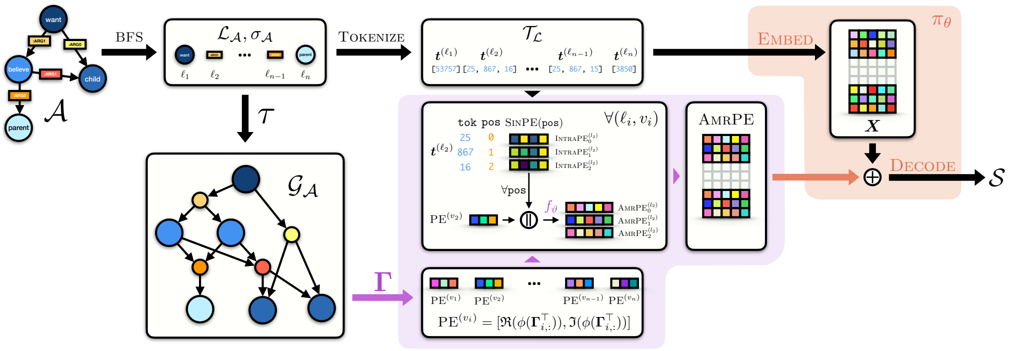


Figure 1: **Overview of SAFT.** An AMR graph $\mathcal{A}$ is first linearized into a token sequence $\mathcal{L}_\mathcal{A}$. We then construct a graph transformation $\mathcal{G}_\mathcal{A}$ and compute structure-aware positional encodings from its magnetic Laplacian. These encodings are combined with standard token positions to form AMR-specific embeddings (AMRPE). A simple MLP f_ϑ aligns them with the embedding space of the LLM π_θ, after which they are injected into the token embeddings X. The model is fine-tuned to generate text $\mathcal{S}$, enabling structure-aware AMR-to-text generation without altering the LLM architecture.

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 Figure 1.

We propose a conceptually general approach for injecting structural inductive bias into LLMs via graph positional encodings. In this work, we specifically evaluate our idea in the context of AMR-to-text generation as it provides a linguistically motivated, semantically rich benchmark that allows for precise evaluation of structural understanding in language generation. SAFT provides a concrete step toward aligning graph structures with pretrained LLMs, focusing on AMRs as a high-value benchmark for studying structure-aware generation, while we recognize that the field of structure-to-text tasks extends beyond AMRs. 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 *structural information* 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 concepts of 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 $\boldsymbol{D}_S$. While this is not *always* necessary, it is required in our approach, as it 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 $\boldsymbol{\Lambda} = \operatorname{diag}(\lambda_1, \ldots, \lambda_{n_\mathcal{G}}) \in \mathbb{R}^{n_\mathcal{G} \times n_\mathcal{G}}$ 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 withing 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 $\boldsymbol{U}$:

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

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

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.

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), 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 extended to AMRs (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). 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). We report a pseudo-code description of our approach in Algorithm 1 and a visualization in Figure 1.

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. This is similar in spirit to the reification used in Semantic Role Labeling or Resource Description Framework graphs (Marcheggiani and Titov, 2017; Marcheggiani and Perez-Beltrachini, 2018), but our transformation is tailored to AMRs and magnetic-Laplacian PEs.

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 $\tau : \Sigma^* \times (\Sigma \rightarrowtail \mathcal{V}_\mathcal{G}) \to \mathbb{G}$ 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 $\phi(v_i) \in \mathbb{C}^k$, $\phi(v_i) = [\boldsymbol{\Gamma}_{i,1}, \ldots, \boldsymbol{\Gamma}_{i,k}]^\top$. We convert $\phi(v_i)$ to a real-valued vector $\mathrm{PE}^{(v_i)} \in \mathbb{R}^{2k}$ by concatenating the real and imaginary parts:

$$\mathrm{PE}^{(v_i)} = [\Re(\phi(v_i)) \; \Im(\phi(v_i))]. \tag{4}$$

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 $v_i = \sigma_{\mathcal{A}}(i)$ we apply sinusoidal positional encodings (Vaswani et al., 2017) to preserve their intra-node ordering:

$$\textsc{IntraPE}_j^{(\ell_i)} = \textsc{SinPE}(j), \quad \text{for } j = 1, \ldots, p_i, \tag{5}$$

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

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

$$\textsc{AmrPE}_j^{(\ell_i)} = f_\vartheta \left(\mathrm{PE}^{(v_i)} \,\|\, \textsc{IntraPE}_j^{(\ell_i)} \right), \tag{6}$$

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 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:

$$\textsc{AmrPE} = \left(\textsc{AmrPE}_1^{(\ell_1)} \ldots \textsc{AmrPE}_{p_1}^{(\ell_1)} \ldots \textsc{AmrPE}_1^{(\ell_2)} \ldots \textsc{AmrPE}_{p_L}^{(\ell_L)} \right)^\top, \tag{7}$$

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. $\textsc{AmrPE}$ 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. 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:

$$\mathcal{T}_{\mathcal{L}} = \boldsymbol{t}^{(\ell_1)} \,\|\, \boldsymbol{t}^{(\ell_2)} \,\|\, \ldots \,\|\, \boldsymbol{t}^{(\ell_L)} = (t_1, \ldots, t_p) \tag{8}$$

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:

$$\boldsymbol{X} = \textsc{Embed}(\mathcal{T}_{\mathcal{L}}) \in \mathbb{R}^{p \times d_{\text{emb}}}. \tag{9}$$

Integrating AMR positional encodings. We then integrate structure-aware positional encodings to the embedded representation of the linearized sequence of tokens through the additive representation

$$\boldsymbol{H} = \boldsymbol{X} + \textsc{AmrPE} \in \mathbb{R}^{p \times d_{\text{emb}}}, \tag{10}$$

which preserves the dimensionality of the LLM embedding space. This modification affects only the input embeddings; the base model's positional encoding mechanism (e.g., RoPE (Su et al., 2024)) remains unchanged and continues to operate inside the attention layers. Finally, $\boldsymbol{H}$ is then fed to the LLM decoder components, including the transformer layers, head, and tokenizer, to return the generated output sequence $\mathcal{S} \in \Sigma^*$, as $\mathcal{S} = \textsc{Decode}(\boldsymbol{H})$.

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 AMR 3.0 benchmark. First, we show that SAFT achieves a new state of the art on sentence-level AMR-to-text generation (Section 4.2). We then demonstrate that it typically improves over conventional fine-tuning, with gains that become more pronounced at increased structural complexity (Section 4.3). We further extend this analysis to document-level AMRs, where SAFT yields even larger improvements (Section 4.4). To separate the effect of fine-tuning from potential pretraining exposure, we also evaluate state-of-the-art closed models in zero-shot and few-shot settings, showing that strong general-purpose LLMs still struggle with AMR-to-text generation without task-specific supervision (Section 4.5). Finally, we analyze how injecting AMRPEs affects the geometry of token representations in the model's latent space (Section 4.6) and report the associated computational overhead (Appendix C.5).

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 pretrained LLMs, 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), with Low-Rank Adaptation (LoRA)[1] (Hu et al., 2022), both with our graph-based positional encoding module (SAFT) and without it (FT). Training and inference settings are identical across conditions. All native components of the base models are preserved; for example, rotary positional embeddings (RoPE) remain active[2]. We report BLEU (Papineni et al., 2002) and chrF (Popović, 2015) scores using greedy decoding, and additionally BERTScore (Zhang et al., 2020) and METEOR (Lavie and Agarwal, 2007) in Appendix C.4. All experiments are implemented in `LitGPT`; further details are in Appendix B.3.

4.2 AMR 3.0 RESULTS

We evaluate SAFT on AMR 3.0 against widely used AMR-to-text baselines: SPRING (Bevilacqua et al., 2021), StructAdapt (Ribeiro et al., 2021), and BiBL (Cheng et al., 2022). Unless explicitly noted, all models are trained on the same AMR 3.0 training split and evaluated on the official held-out test set. SPRING and BiBL additionally report variants trained with extra heuristically labeled data, which we gray out in Table 1. Both models rely on linearized AMR inputs and sequence-to-sequence training. StructAdapt, in contrast, introduces a dedicated graph encoder within an encoder–decoder architecture. Our approach differs in that it operates on modern decoder-only LLMs and injects structural information through graph positional encodings without introducing a separate encoder component. The results in Table 1 present a comparative evaluation of prior approaches[3]

---

[1] SAFT is compatible with any parameter-efficient tuning method.

[2] AMRPEs do not alter or replace the LLM's own positional encoding.

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

(Bevilacqua et al., 2021; Cheng et al., 2022; Ribeiro et al., 2021), alongside our fine-tuned decoder-only LLMs, both with and without the proposed graph positional encoding module (SAFT). While these architectures are not directly matched, this comparison provides useful context for situating decoder-only models within existing AMR-to-text work. 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 typically outperforms FT, with aggregate BLEU gains of +0.8 over the FT variant across the different models. However, such aggregate scores can obscure important variation across inputs of different structural complexity. To more accurately characterize when and how SAFT helps, we turn to a stratified analysis.

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:

$$\Delta_{\text{BLEU}}(z) = \text{BLEU}^z_{\text{SAFT}} - \text{BLEU}^z_{\text{FT}}, \quad (11)$$

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 $\delta(\mathcal{A}) \geq z$. Figure 2a 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.

The divergence of the curves at greater depths and an overall upward trend across all models highlight the increasing importance of modeling structure explicitly as semantic complexity grows. While Gemma 2B showed little overall improvement across the full dataset (see Table 1), the benefit of SAFT becomes more pronounced on examples with greater semantic complexity.

Table 1: **SAFT achieves state-of-the-art results on AMR-to-text generation.** We report BLEU/CHRF on AMR 3.0, comparing: prior work, and 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. We report SAFT (ours) relative improvement over FT in brackets.

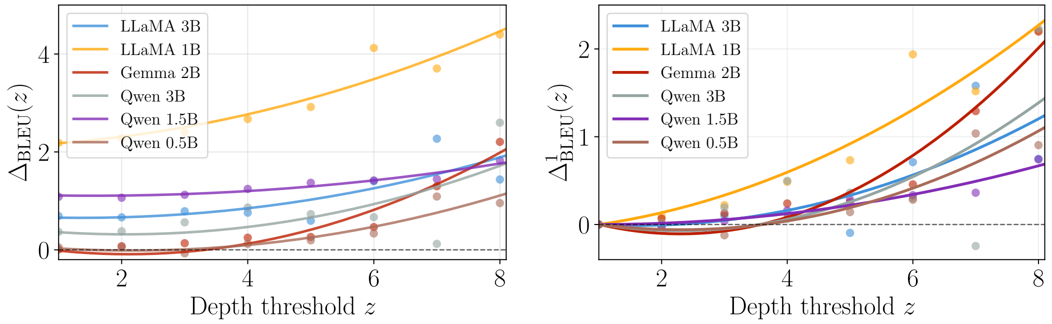
Model	**Variant**	**BLEU ↑**	**CHRF ↑**
Previous Work			
BiBL	w/o Extra data	47.4	74.5
	+ Extra data	50.7	76.7
SPRING	w/o Extra data	44.9	72.9
	+ Extra data	46.5	73.9
StructAdapt	w/o Extra data	48.0	73.2
Our Finetuned LLMs: trained without extra data			
LLaMA 3.2 (3B)	FT	53.5	75.5
	SAFT	**54.2** (+1.3%)	**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).

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:

$$\Delta^1_{\text{BLEU}}(z) = \Delta_{\text{BLEU}}(z) - \Delta_{\text{BLEU}}(1). \quad (12)$$

Figure 2b 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.



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

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

Figure 2: **BLEU score improvements** of structurally-aware fine-tuned (SAFT) models over conventionally fine-tuned (FT) counterparts, on AMRs of depth $\delta(\mathcal{A}) \geq z$. (a) **Absolute improvement** (Δ_{BLEU}): differences in BLEU between SAFT and FT models across graph depths and model families. (b) **Relative improvement** (Δ^1_{BLEU}): differences in BLEU between SAFT and FT models across graph depths and model families normalized by performance at depth-1 graphs. Both plots reveal an increasing advantage of SAFT as structural complexity grows, demonstrating its effectiveness in leveraging graph topology for improved generation. Lines are 2nd-degree polynomial fits.

4.4 DOCAMR RESULTS

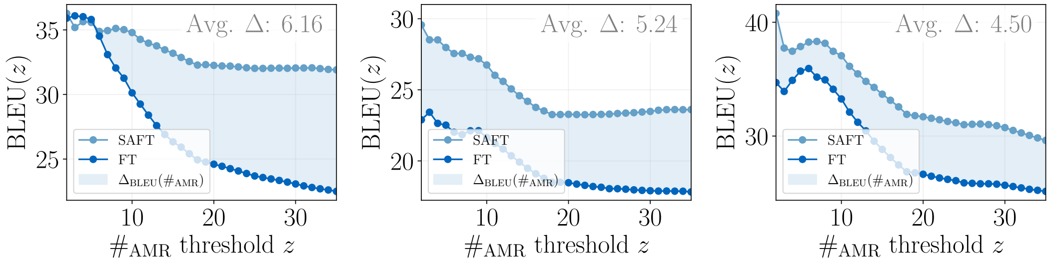
To understand even further how SAFT impacts performance on highly complex AMRs, we evaluate both SAFT and standard fine-tuning (FT) on DOCAMR, a supplementary subset of AMR 3.0 (Knight et al., 2020) whose test set consists of sentence-level AMRs from the standard AMR test split, merged into documents 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 cross-dataset generalization capabilities 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 Figure 3, SAFT consistently improves performance across all levels of complexity which is measured by $\#_{\mathrm{AMR}}$, 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. SAFT, 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.

4.5 EVALUATION UNDER ZERO-SHOT AND FEW-SHOT PROMPTING ON SOTA LLMS

We additionally evaluate two closed-source large language models, GPT-4o-mini and GPT-4o, in both zero-shot and few-shot settings. We use a fixed task prompt with seven AMR–text examples; the full template is provided in Appendix B.3.2. As shown in Table 2, moving from zero-shot to few-shot leads to only marginal improvements for both models. Despite their scale, their performance remains substantially below that of a much smaller 3B model fine-tuned on AMR. In fact, standard



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

Figure 3: **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.

Table 2: **Comparison of zero-shot, few-shot, and finetuned models.** We report BLEU, CHRF, BERTScore, and METEOR. Best results per metric are bolded.

Model	**Setting**	**BLEU ↑**	**CHRF ↑**	**METEOR ↑**	**BERTScore ↑**
GPT-4o-mini	Zero-shot	16.3	44.8	38.4	74.8
GPT-4o-mini	Few-shot	17.0	45.2	39.9	75.7
GPT-4o	Zero-shot	28.0	59.8	52.1	81.1
GPT-4o	Few-shot	29.6	60.1	52.4	81.8
Qwen 2.5 (3B)	FT	51.6	72.1	59.4	82.9
	SAFT (ours)	**51.9**	**74.8**	**60.1**	**83.7**

fine-tuning (FT) of Qwen 2.5 (3B) already more than doubles the BLEU score of GPT-4o-mini, and SAFT further improves on FT using the same backbone. This suggests that AMR-to-text generation is not easily solved through prompting alone under our prompting setup. In-context examples provide limited signal about the underlying graph structure, whereas fine-tuning exposes the model to the full supervision of the training set and allows SAFT to inject explicit structural information directly.

4.6 GEOMETRIC ANALYSIS OF AMRPES

We investigate how the integration of structural information via our AMRPEs (Eq. (7)) modifies the token embeddings $\boldsymbol{X}$, generating the representations $\boldsymbol{H} = \boldsymbol{X} + \text{AMRPE}$ (Eq. (10)). Figure 4 summarizes our results. **(a) Norm in the same order of magnitude.** AMRPEs have a larger norm than token embeddings, with a median ratio of $\|\text{AMRPE}\|_2 / \|\boldsymbol{X}\|_2 \approx 3$, but remain within the same order of magnitude. This implies that the structural signal is not obscured by the text token embeddings, nor does it obscure them in turn. **(b) Orthogonal direction.** The cosine between AMRPE and $\boldsymbol{X}$ is concentrated around zero, indicating that structural embeddings add information about directions that are generally orthogonal to the textual subspace. This suggests an enrichment rather than an overwriting of semantic content. **(c) Variance concentrated on a few axes.** Projecting AMRPEs in the PCA basis of $\boldsymbol{X}$ reveals that most of its variance is concentrated on a small number of principal directions, with a single dominant component carrying most of the information. The structural signal is therefore highly directional rather than diffuse. **(d) Consistent shift.** Projecting $\boldsymbol{X}$ and $\boldsymbol{H}$ into the PCA space of $\boldsymbol{X}$, we observe that AMRPEs induce a consistent and directed shift of $\boldsymbol{X}$ along a few dominant directions, with the residual variance distributed over multiple weaker components. **(e) Greater intrinsic dimensionality.** The spectrum of explained variance of $\boldsymbol{H}$ is flatter than that of $\boldsymbol{X}$, requiring many more components to achieve the same variance thresholds. This reflects a clear increase in intrinsic dimensionality. Overall, this geometric separation suggests a potential mechanism by which SAFT may avoid interfering with semantic representations, though direct evaluation of broader capabilities is left for future work.

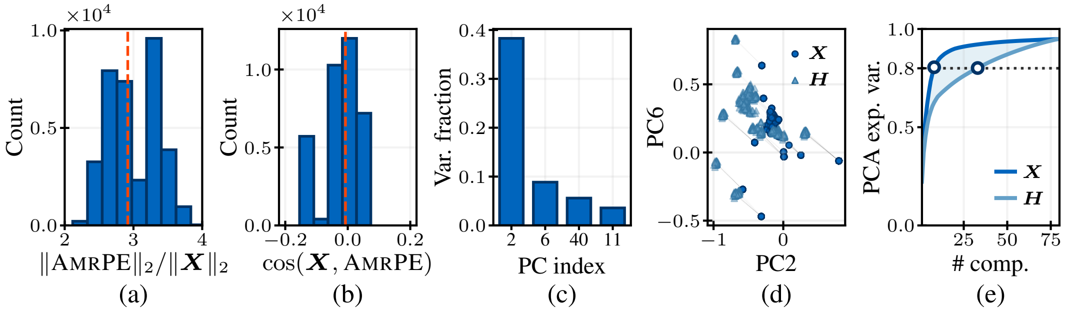


Figure 4: **Effect of AMR positional embeddings (AMRPE) on token representation geometry.** (a) AMRPE has a larger but same-order magnitude $\boldsymbol{X}$. (b) It is injected along directions largely orthogonal to $\boldsymbol{X}$. (c) Its variance concentrates on a small number of PCA directions of $\boldsymbol{X}$. (d) In this PCA space, $\boldsymbol{H} = \boldsymbol{X} + \text{AMRPE}$ shifts coherently along the dominant axis. (e) $\boldsymbol{H}$ exhibits a flatter explained-variance spectrum, indicating increased intrinsic dimensionality.

5 RELATED WORK

To the best of our knowledge, SAFT is the first work to inject graph positional encodings into LLMs for structure-aware fine-tuning without modifying the model's architecture. SAFT is architecture-agnostic: it introduces relational inductive bias through precomputed, parameter-free encodings (from the magnetic Laplacian spectrum), injected via a lightweight projection into the LLM's embedding space. This eliminates the need for graph-specific training while allowing direct fine-tuning of the LLM. Prior work on graph-to-text generation, particularly AMR-to-text, can be grouped into three families: **linearization-based**, **adapter-based**, and **graph-tuning-based**. We discuss here some of the main approaches in each family and report additional details in Appendix F.

Linearization-based. Graphs are serialized into sequences for seq2seq models such as BART or T5. Examples include SPRING (Bevilacqua et al., 2021), AMR-BART (Bai et al., 2022), and BiBL (Cheng et al., 2022), which differ in traversal strategies and auxiliary tasks. LLMs have also been adapted via fine-tuning (Raut et al., 2025; Mager et al., 2020) or prompting (Yao et al., 2024a; Jin et al., 2024b). Similar ideas extend beyond AMR to molecular graphs (Zheng et al., 2024), tables (Fang et al., 2024), and 3D meshes (Wang et al., 2024).

Adapter-based. These approaches introduce graph-native components, such as GNN-based adapters or modified attention layers, to inject relational information. StructAdapt (Ribeiro et al., 2021) uses graph-aware adapters within pretrained transformers, while others directly encode AMR graphs via graph-to-sequence models (Zhu et al., 2019; Song et al., 2018; Wang et al., 2020).

Graph-tuning-based. Recent methods integrate learned graph modules with LLMs, for example by training GNN-based adapters (Huang et al., 2024), adding graph transformers at each layer (Chai et al., 2023), or prepending GNN-derived embeddings into the prompt (Tang et al., 2024). While effective, these strategies require additional trainable modules, architectural changes, or costly pretraining.

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.

REPRODUCIBILITY STATEMENT

We have taken deliberate steps to ensure the reproducibility of our results. The implementation of SAFT builds directly on the open-source `LitGPT` framework, with all modifications described in Appendix B.3 and detailed pseudo-code in Appendix B.2. Hyperparameter configurations, training schedules, and model-specific settings are reported in Appendix B.3 and Tables 4 and 5. Datasets (AMR 3.0 and DocAMR) are described in Appendix E.1, including licensing details. Our evaluation uses standard metrics (BLEU, ChrF, BERTScore, METEOR), with references provided in Section 4 and Appendix C.4. Computational overhead analyses are included in Appendix C.5, and zero-shot baselines are reported in Section 4.5.

USE OF LARGE LANGUAGE MODELS

Large language models (LLMs) are a central component of our proposed method: we fine-tune pretrained LLMs as the backbone of our approach. In addition, LLMs were used as a writing aid to polish the text of the paper, improving clarity, grammar, and style. LLMs were not used to generate research ideas, design experiments, analyze results, or draw conclusions. All conceptual and scientific contributions are the responsibility of the authors.

REFERENCES

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.

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.

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.

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.

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*, page 491–500, New York, NY, USA, 2024. Association for Computing Machinery.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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

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

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.

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

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.

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

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.

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.

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.

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.

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.

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.

Diego Marcheggiani and Ivan Titov. Encoding Sentences with Graph Convolutional Networks for Semantic Role Labeling. In Martha Palmer, Rebecca Hwa, and Sebastian Riedel, editors, *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing*, pages 1506–1515, Copenhagen, Denmark, September 2017. Association for Computational Linguistics.

Diego Marcheggiani and Laura Perez-Beltrachini. Deep Graph Convolutional Encoders for Structured Data to Text Generation. In Emiel Krahmer, Albert Gatt, and Martijn Goudbeek, editors, *Proceedings of the 11th International Conference on Natural Language Generation*, pages 1–9, Tilburg University, The Netherlands, November 2018. Association for Computational Linguistics.

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

Jianlin Su, Murtadha Ahmed, Yu Lu, Shengfeng Pan, Wen Bo, and Yunfeng Liu. Roformer: Enhanced transformer with rotary position embedding. *Neurocomput.*, 568(C), February 2024.

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.

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.

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.

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.

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.

Tianyi Zhang, Varsha Kishore, Felix Wu, Kilian Q. Weinberger, and Yoav Artzi. Bertscore: Evaluating text generation with bert, 2020. URL https://arxiv.org/abs/1904.09675.

Alon Lavie and Abhaya Agarwal. Meteor: an automatic metric for mt evaluation with high levels of correlation with human judgments. In *Proceedings of the Second Workshop on Statistical Machine Translation*, StatMT '07, page 228–231, USA, 2007. Association for Computational Linguistics.

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.

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.

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.

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.

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.

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.

Xuanwen Huang, Kaiqiao Han, Yang Yang, Dezheng Bao, Quanjin Tao, Ziwei Chai, and Qi Zhu. Can gnn be good adapter for llms? In *Proceedings of the ACM Web Conference 2024*, page 893–904, New York, NY, USA, 2024. Association for Computing Machinery.

Ziwei Chai, Tianjie Zhang, Liang Wu, Kaiqiao Han, Xiaohai Hu, Xuanwen Huang, and Yang Yang. Graphllm: Boosting graph reasoning ability of large language model. *arXiv preprint arXiv:2310.05845*, 2023.

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.

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.

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.

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.

APPENDIX

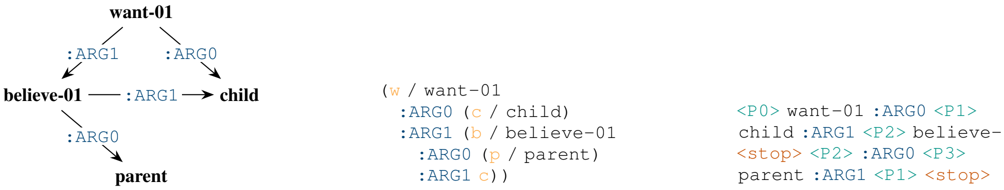
A ABSTRACT MEANING REPRESENTATION

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.*

Figure 5a is the graph representation of the sentence, while Figures 5b and 5c are the Penman and BFS linearizations, respectively.



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

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

(a) Graph representation (b) Penman notation (c) BFS linearization

Figure 5: 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.2 VISUALIZING OTHER AMR EXAMPLES

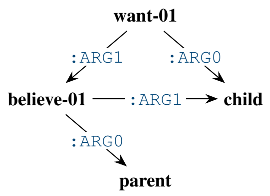
We visualize the linearization and preprocessing steps, as detailed in Appendix B.1 for additional sentences. Table 3 summarizes the examples and points to the corresponding figures showing the original AMR graphs and the transformation pipeline.

Table 3: Examples with sentences, AMR graphs, and preprocessing steps

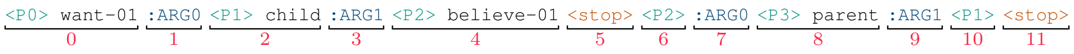
Ex.	Sentence	Original AMR Graph	Preprocessing Steps
I	I used to play tennis.	Figure 11	Figure 12
II	This is really eye-opening.	Figure 13	Figure 14
III	The key is to be as objective as possible.	Figure 15	Figure 16
IV	Speeding and accidents have surged as well.	Figure 17	Figure 18

B IMPLEMENTATION DETAILS

B.1 SEMANTICALLY-PRESERVING AMR TRANSFORMATION

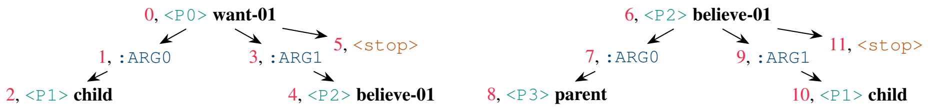


(a) Input AMR graph $\mathcal{A}$

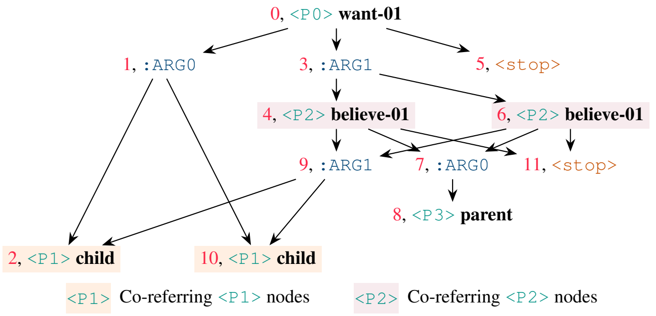


(b) $\mathcal{L}_\mathcal{A} = (\ell_1, \dots, \ell_L) = \text{BFS}(\mathcal{A})$, $\sigma_\mathcal{A} : \mathbb{Z} \rightarrowtail \mathcal{V}_\mathcal{G}$

(c) ToSubgraph



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



(e) Merge

Figure 6: Transformation steps from an AMR graph $\mathcal{A}$ to the Role-Expanded Graph $\mathcal{G}$.

We report detailed information of our proposed semantically-preserving transformation $\tau : \Sigma^* \times (\Sigma \rightarrowtail \mathcal{V}_\mathcal{G}) \to \mathbb{G}$ 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 Figure 6b—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, Figure 6c): $\mathcal{L}_\mathcal{A}$ 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, Figure 6d): Each labeled edge $(u \xrightarrow{r} v)$ is expanded into a role node r, creating two unlabeled edges: $(u \to r)$ and $(r \to v)$. 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, Figure 6d): 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** ($\sigma_\mathcal{A}^{-1}$, Figure 6d): 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, Figure 6e): For each pointer index j (e.g., `<P2>`), identify co-referring nodes $\mathcal{U}_j = \{u_1, \dots, u_k\}$ 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 $\mathcal{E}^{\text{in}}(u_i) = \{(v, u_i) : (v, u_i) \in \mathcal{E}_\mathcal{A}\}$.
    - (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 $u_i \in \mathcal{U}_j$: $\mathcal{E}^{\text{in}}(u_i) := \mathcal{E}^{\text{in}}_{\mathcal{U}_j}, \quad \mathcal{E}^{\text{out}}(u_i) := \mathcal{E}^{\text{out}}_{\mathcal{U}_j}$

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

B.2 ALGORITHM

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))
    $\phi(v_i) = \mathbf{\Gamma}_{i,:}^\top$    ▷ Select complex eigenvector
    $\text{PE}^{(v_i)} = [\Re(\phi(v_i)) \; \Im(\phi(v_i))]$    ▷ Node-level PE (Eq. (4))
    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. (5))
        $\textsc{AmrPE}_j^{(\ell_i)} = f_\vartheta(\text{PE}^{(v_i)} \,\|\, \textsc{IntraPE}_j^{(\ell_i)})$    ▷ Token-wise AMR PE (Eq. (6))
    end for
end for
$\textsc{AmrPE} = (\textsc{AmrPE}_1^{(\ell_1)} \dots \textsc{AmrPE}_{p_L}^{(\ell_L)})^\top$    ▷ AMR PE (Eq. (7))
$\mathcal{T}_\mathcal{L} = \boldsymbol{t}^{(\ell_1)} \,\|\, \boldsymbol{t}^{(\ell_2)} \,\|\, \dots \,\|\, \boldsymbol{t}^{(\ell_L)}$    ▷ Token sequence (Eq. (8))
$\boldsymbol{X} = \textsc{Embed}(\mathcal{T}_\mathcal{L})$    ▷ Token sequence embedding
$\boldsymbol{H} = \boldsymbol{X} + \textsc{AmrPE}$    ▷ Inject structure-aware PE (Eq. (10))
$\mathcal{S} = \textsc{Decode}(\boldsymbol{H})$    ▷ Generate output sequence
```

B.3 TRAINING DETAILS

Training setup. We build on the open-source `LitGPT`[4] framework and extend it to incorporate our structure-aware representations. In particular, we (i) generate graph-based positional encodings (AMRPE), (ii) align AMR nodes to token positions via node-aware tokenization, (iii) introduce a lightweight projection layer f_θ to map these encodings into the LLM's embedding space, and (iv) add task-specific prompting to support AMR-to-text generation.

As in standard sequence-to-sequence fine-tuning, we optimize a cross-entropy objective. The pre-trained model weights are updated only through LoRA adapters, while the parameters of f_θ are trained from scratch. All other LLM parameters remain frozen.

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.3.1 HYPERPARAMETERS

The hyperparameter choice for each model can be found in Table 4 and Table 5.

Table 4: Hyperparameter configurations for each SAFT models

Category	Hyperparameter	LLaMA 1B	LLaMA 3B	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 5: Hyperparameter configurations for each conventionally fine-tuned model.

Category	Hyperparameter	LLaMA 1B	LLaMA 3B	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,

---

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

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, $\text{LR}_{f_\theta}(t) = \mu \cdot \text{LR}(t)$, where $\text{LR}(t)$ 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.

B.3.2 PROMPTING FORMAT

Fine-tuning prompt. To enable AMR-to-text generation with 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:

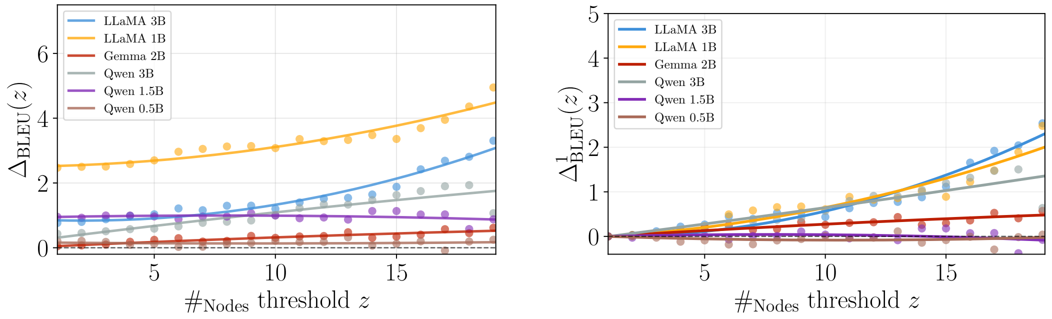
```
    [Output:  Text]
```

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]
```

Few-shot prompt for GPT models. The full prompt passed to the model is structured as:

```
You are an assistant that converts AMRs to fluent English
sentences.
Your job:  For each item, convert its ‘amr‘ into one fluent
English sentence.
Do not include explanations; only output JSON.

Output format (STRICT):
{
"predictions":  [ { "id":  "...", "text":  "..." } ]
}

### FEW-SHOT EXAMPLES

Example 1 (input):  ["id":"1","amr":"(u/understand-01 :ARG0
(i/i)
:ARG1 (t/thing :ARG1-of (s/say-01 :ARG0 (p/person :name
(n/name :op1 "Ron" :op2 "Paul"))))"]
Example 1 (output):  {"predictions":[{"id":"1","text":"I get
what Ron Paul is saying."}]}

Example 2 (input):  ["id":"2","amr":"(s/say-01 :ARG0
(l/libertarian
:mod (h/hardcore)) :ARG1 (t/that))"]
Example 2 (output):  {"predictions":[{"id":"2","text":"Thats
what a Hardcore
Libertarian would say."}]}

Example 3 (input):  ["id":"3","amr":"(h/hard-02 :degree
(k/kind-of) :ARG1
(i/import-01 :ARG1 (f/food)) :ARG1-of (c/cause-01 :ARG0
(c2/consider-01
:ARG1 (p/probable :domain (w/wipe-out-02 :ARG1 (s/store)
:mod (t/too))))))"]
Example 3 (output):  {"predictions":[{"id":"3","text":"Kind
of hard to import food
considering that the stores are probably wiped out too."}]}

...

### NOW PROCESS THE REAL BATCH
[ {"id":"0","amr":"(d/date-entity :day 21 :month 8 :year
2007)"} ]
```



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

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

Figure 7: 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 Figure 2), indicating that structural complexity plays a more critical role than graph size alone. All lines are second-degree polynomial fits.

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 influence the performance of SAFT. As shown in Figure 7, SAFT exhibits consistent improvements over standard fine-tuning across most model sizes, particularly for larger models. However, the trend is less pronounced than in Figure 2, 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.

C.2 EFFECT OF MODEL SCALE ON SAFT

A common hypothesis is that structural inductive biases become less relevant as model scale increases, under the assumption that sufficiently large language models can internalize structural reasoning through parametric capacity alone. Our results do not provide evidence in support of this hypothesis within the range of model sizes we evaluate.

As shown in Figure 8, the relative improvements of SAFT over standard fine-tuning (FT) do not exhibit a monotonic decreasing trend with increasing model size. Instead, gains fluctuate across both scale and model family. For example, the largest BLEU improvements are observed for mid-sized LLaMA models, while the strongest ChrF improvements occur for the 3B Qwen model. Moreover, these trends are not consistent across metrics: within a single model family, BLEU and ChrF gains follow different trajectories as scale increases, and model rankings vary depending on the evaluation measure.

We emphasize that this does not rule out the possibility that, at larger scales than those considered in this study, the benefits of SAFT may eventually diminish or disappear. However, within the regimes we test, no such pattern is observable. Instead, the effect of SAFT appears to depend on a combination of model scale, architecture, and generation dynamics, rather than scale alone. Therefore, based on current evidence, the claim that increasing model size renders SAFT unnecessary is not supported.

C.3 BOOTSTRAP PAIRED SIGNIFICANCE TEST

To evaluate whether the performance differences between SAFT and standard fine-tuning (FT) are robust to test-set variation, we conduct a bootstrap paired significance analysis. For each model, we repeatedly resample the AMR 3.0 test set with replacement and compute BLEU, and ChrF for both FT and SAFT on each bootstrap replicate. This yields paired score distributions without requiring multiple independent training runs, allowing us to isolate test-set variability as a source of uncertainty.

Table 6 reports the bootstrap mean and standard deviation for each metric and model. Across all settings, the variance induced by resampling is small (typically within ±0.3–0.5 BLEU), indicating that the observed performance differences are not driven by unstable test-set fluctuations.

To further characterize the consistency of these differences, Figure 9 presents a win-rate matrix, showing for each model–metric pair the fraction of bootstrap samples in which SAFT outperforms FT. In most cases, SAFT wins on a clear majority of samples, including all LLaMA variants and two out of the three Qwen models. For settings where the average improvements are small, the bootstrap distributions reflect this appropriately through near-balanced win rates, rather than artificially inflating significance.

Overall, this analysis indicates that the gains reported in the main results are stable under reasonable test-set perturbations. At the same time, the bootstrap results remain appropriately conservative in cases where the FT–SAFT gap is minimal, avoiding overinterpretation of marginal differences.

C.4 EXTRA EVALUATION METRICS

Along with the metrics reported in Section 4.2, we evaluate SAFT vs. FT using METEOR (Lavie and Agarwal, 2007) and BERTScore (Zhang et al., 2020), two complementary metrics that are more sensitive to semantic adequacy and fluency. The results are shown in Table 7. We further stratified the gains by AMR graph depth, following the same setup as in Figure 2. Both BERTScore and METEOR gains increase with graph depth as shown in Table 8. These results support our claim: SAFT becomes increasingly beneficial for structurally complex inputs, and these improvements hold across multiple metrics.

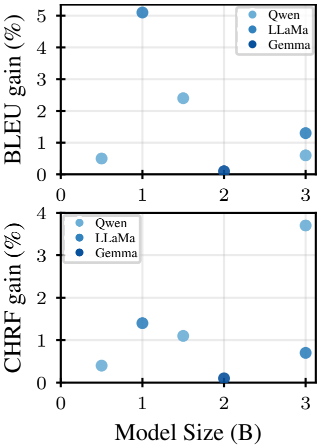


Figure 8: **Relative BLEU and ChrF++ improvements as a function of model size.**

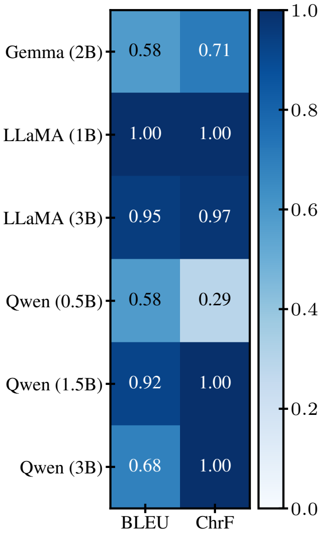


Figure 9: **Win-rate heatmap of the bootstrap paired test.**

C.5 RUNTIME

We provide a formal breakdown of SAFT's time and space complexity and clarify the scope of its computational overhead. All graph-related operations are performed once at preprocessing time and do not affect training or inference time. These include: graph preprocessing, magnetic Laplacian computation, partial eigendecomposition (EVD), and positional encoding computation. The main bottleneck in this process is the partial EVD.

The dense partial EVD requires $\mathcal{O}(kn^2)$, where k is the number of computed eigenvalues, n the number of nodes. For most practical scenarios in the AMR-to-text generation task, this complexity is manageable, as AMRs are relatively small, with AMR 3.0 ($\sim$54 nodes) and DocAMR ($\sim$730 nodes). Asymptotically, sparse solvers become advantageous when $n \gg k$. This translates in practice to $n \gtrsim 2000$. In this case, the complexity is $\mathcal{O}(km)$ where m is the number of edges.

We first show in Table 9 that, in our case, using dense solvers yields faster runtime compared to sparse solvers. Then, we report in Table 10 the average and maximum time (in seconds) required for inference and precomputation, showing the negligible impact of the latter. The computation of

Table 6: **Bootstrap means and standard deviations of BLEU, and ChrF over 50 resampled test sets.** The variability is small across models, and the relative behavior of SAFT vs. FT aligns with the trends reported in the main results. Bold results are statistically significant ($p < 0.05$).

Model	**Variant**	**BLEU**	**ChrF**
Gemma (2B)	FT	52.97 ± 0.49	69.74 ± 0.29
	SAFT	53.01 ± 0.48	69.98 ± 0.30
LLaMA 3.2 (1B)	FT	45.64 ± 0.55	65.74 ± 0.37
	SAFT	**47.75 ± 0.54**	**67.65 ± 0.33**
LLaMA 3.2 (3B)	FT	53.52 ± 0.40	71.70 ± 0.25
	SAFT	54.45 ± 0.44	72.40 ± 0.30
Qwen 2.5 (0.5B)	FT	42.68 ± 0.40	63.14 ± 0.30
	SAFT	42.94 ± 0.52	62.94 ± 0.30
Qwen 2.5 (1.5B)	FT	50.66 ± 0.46	68.69 ± 0.30
	SAFT	51.69 ± 0.54	**70.70 ± 0.34**
Qwen 2.5 (3B)	FT	51.75 ± 0.50	69.33 ± 0.37
	SAFT	52.00 ± 0.45	**70.44 ± 0.29**

Table 7: **Comparison of FT vs. SAFT.** We report METEOR and BERTScore for Qwen 2.5 and LLaMA 3.2 (3B) on AMR 3.0. Best results per metric are highlighted in bold, with relative gain in parentheses.

Model	**Variant**	**METEOR ↑**	**BERTScore ↑**
Qwen 2.5 (3B)	FT	59.4	82.85
	SAFT	**60.1** (+1.18%)	**83.69** (+1.01%)
LLaMA 3.2 (3B)	FT	70.3	86.19
	SAFT	**70.5** (+0.28%)	**86.22** (+0.03%)

Table 8: **Impact of graph depth on gains.** We report relative improvements in BERTScore and METEOR across different depths.

Graph Depth	**BERTScore Gain ↑**	**METEOR Gain ↑**
0	0.84	1.08
3	1.30	1.31
9	3.11	5.01
10	2.96	3.27

our structure-aware encodings is a preprocessing step that leaves the model architecture and runtime efficiency intact. Since structure-aware encodings are computed once per graph and reused throughout training and inference, the overall overhead remains minimal.

At inference time, SAFT does not change the computation inside the Transformer itself. The token sequence length and the hidden dimensionality remain the same, so the attention and feed-forward layers have the same cost as in standard fine-tuning (FT). The only additional costs compared to FT come from two sources. First, we compute the graph positional encodings for the input AMR. This is done once per graph and does not depend on the decoding length. As shown in Table 10, this step adds on average 0.01 seconds on AMR 3.0 and 0.28 seconds on DocAMR, which is small compared to the overall inference time. Second, we apply a lightweight two-layer MLP to project these encodings into the model's embedding space. This MLP maps $\mathbb{R}^{2k+d} \to \mathbb{R}^{d_{\text{emb}}}$ and $\mathbb{R}^{d_{\text{emb}}} \to \mathbb{R}^{d_{\text{emb}}}$, and introduces $d_{\text{emb}}(2k + d + d_{\text{emb}} + 2)$ additional parameters. This overhead is negligible relative to the size of the underlying LLM.

C.6 POTENTIAL APPLICABILITY BEYOND AMR

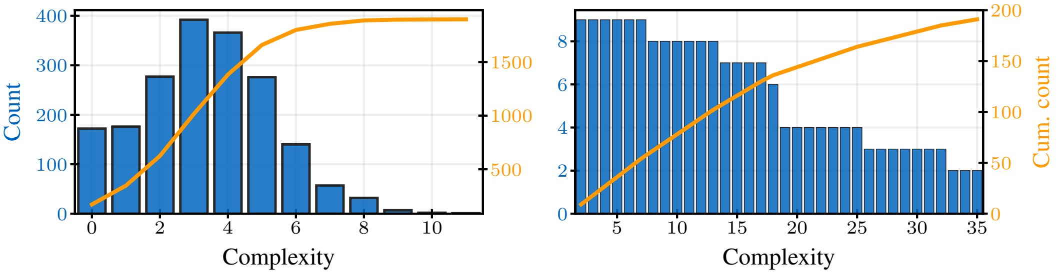
SAFT does not assume properties unique to AMR and only requires a graph structure together with a node-to-token alignment. This suggests a potential extensions to other graph-structured data associated with text, such as semantic role graphs, knowledge graph triplets, or discourse trees mapped to sentences, just to name a few. A full empirical study of these settings is outside the scope of this work, but we view this as a promising direction for future research.

Table 9: **Sparse vs. dense preprocessing times.** We report average and maximum runtimes (in seconds) for AMR 3.0 and DocAMR.

Dataset	Method	Avg Time (s) ↓	Max Time (s) ↓
AMR 3.0	Sparse	0.009	0.12
	Dense	0.027	0.57
DocAMR	Sparse	1.31	12.51
	Dense	0.28	2.49

Table 10: **Inference and preprocessing times.** Inference time and graph preprocessing time (including EVD, sinusoidal encoding, and projection via the MLP) for Qwen 3B.

Dataset	Inference Time (s)		Preprocessing Time (s)	
AMR 3.0	Avg: 1.73	Max: 6.81	Avg: 0.01	Max: 0.12
DocAMR	Avg: 325.24	Max: 582.22	Avg: 0.28	Max: 2.49



(a) AMR 3.0

(b) DocAMR

Figure 10: Count of AMR graphs per complexity on both AMR 3.0 and DocAMR (test sets).

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 encoding 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[5]) (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.

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. DocAMR consists of 284 documents in the training split and 9 documents in the test split, covering a total of 8027 gold sentence-level AMRs.

---

[5] `https://catalog.ldc.upenn.edu/LDC2020T02`

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.

To make sample availability across structural complexities explicit, we include histograms of graph depths for AMR 3.0 and DocAMR (Figure 10). AMR 3.0 exhibits a long tail with few very high-depth graphs, whereas DocAMR contains nine document-level graphs whose constituent sentence-level AMRs all originate from the AMR 3.0 test split, each covering many AMRs. To test the capabilities of our model across multiple complexities (number of AMRs in the case of DocAMR), we sample subgraphs from document-level AMRs of increasing complexities, with a resulting count distribution as shown in Figure 10(b).

Table 11: 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	284 / - / 9	Document-level annotations, coreference, discourse	LDC Non-Member License

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[6], an open-source project released under the Apache License 2.0[7]. It provides modular components for efficient fine-tuning, inference, and reproducibility across large-scale models.

Qwen. Qwen models[8], 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[9], 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[10], released by Meta, are governed by the LLAMA 2 Community License Agreement. The license permits use, modification, and redistribution, but restricts:

- Commercial use by entities exceeding 700 million monthly active users without explicit permission from Meta
- 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[11].

---

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

Table 12: 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

F ADDITIONAL INFORMATION ON RELATED WORK

Prior work on AMR-to-text generation—and more broadly, text generation from graph-structured data—has been explored through three main paradigms: **linearization-based approaches**, which serialize graphs into sequences; **adapter-based approaches**, which introduce graph-native modules into pretrained architectures; and **graph-tuning-based approaches**, which couple LLMs with trainable graph encoders or embeddings. To the best of our knowledge, no prior work has investigated graph positional encodings as a lightweight, architecture-agnostic means of enabling structure-aware fine-tuning.

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

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) is built around the encoder–decoder architecture (e.g., T5) in which a dedicated graph encoder (which they call StructAdapt) produces structure-enriched representations that the decoder then consumes via cross-attention and MLP adapters. Its core design relies on this asymmetry: graph processing happens entirely in the encoder, while the decoder only receives those enriched encoder states. 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.

F.4 GRAPH-TUNING-BASED APPROACHES

A more recent line of work integrates graph modules directly into LLMs, aiming to couple structural encoding with pretrained language models. GraphAdapter (Huang et al., 2024) employs a trainable GNN as an adapter, aligned with the LLM during fine-tuning, requiring both pretraining and parameter-intensive alignment. GraphLLM (Chai et al., 2023) inserts graph transformers at every layer of the LLM by introducing learned graph-based prefix tokens into the key/value projections. GraphGPT (Tang et al., 2024) prepends graph embeddings produced by a trainable GNN into the prompt via a learned projector, keeping the LLM frozen. While these methods inject structural information effectively, they introduce substantial complexity in the form of additional trainable modules, architectural modifications, or expensive pretraining requirements.

F.5 COMPARISON AND POSITIONING OF SAFT

Across these paradigms, a common trade-off emerges: linearization-based methods retain compatibility with standard architectures but discard explicit structure; adapter-based methods preserve structure but sacrifice full LLM compatibility; graph-tuning-based methods tightly integrate graphs with LLMs but at the cost of additional parameters and architectural changes. In contrast, SAFT is architecture-agnostic and parameter-efficient. It introduces relational inductive bias through precomputed, parameter-free positional encodings derived from the magnetic Laplacian spectrum. These encodings are aligned with the LLM's embedding space via a lightweight projection layer, requiring no graph-specific training, no architectural changes, and no costly pretraining. This design makes SAFT a simple and efficient way to enable structure-aware fine-tuning of LLMs, while remaining general to tasks involving graph-structured inputs.

G GENERATED OUTPUTS AND ERROR ANALYSIS

We present a qualitative error analysis comparing baseline fine-tuning (FT) and our structure-aware fine-tuning method (SAFT). To avoid cherry-picking and to ensure that examples are sampled in a principled way, we compute smoothed sentence-level BLEU scores for both models on every test instance. We then partition the dataset into performance buckets based on percentile thresholds: cases where SAFT is considerably better than FT, cases where FT is considerably better than SAFT, and cases where both models either perform well or fail. From each bucket, we draw a small random sample of five examples. For each selected instance, we report the reference output, the two model predictions, and their respective BLEU scores. We additionally highlight token-level differences using a color-coded character diff to make divergences visually salient. The goal is not to claim statistical significance from individual examples, but to give the reader concrete insight into the characteristic strengths and failure modes of each system across distinct performance regimes.

SAFT BETTER

Ground Truth: International; weapons; proliferation; Government; energy
FT Prediction: International; weapons; proliferation; Governmentergy; energy [BLEU: 28.6]
SAFT Prediction: International; weapons; proliferation; Government; energy [BLEU: 100.0]

Ground Truth: International; crime; Government; narcotics
FT Prediction: International; crime; gGovernment; narcotics [BLEU: 18.8]
SAFT Prediction: International; crime; Government; narcotics [BLEU: 100.0]

Ground Truth: The issues have been unresolved for 4 years.
FT Prediction: The issues werhave beenot unresolved for four4 years. [BLEU: 7.0]
SAFT Prediction: The issues have been unresolved for 4 years. [BLEU: 70.7]

Ground Truth: You can't get her sectioned for that.
FT Prediction: You can't get aher sectioned forbecausebecause that. [BLEU: 14.8]
SAFT Prediction: You can't get her sectioned for that. [BLEU: 100.0]

Ground Truth: 2. Create a few nuclear-powered aircraft carrier battle groups.
FT Prediction: 2. Createing a few nuclear-powered aircraft carriers groupsbattle groups. [BLEU: 5.6]
SAFT Prediction: 2. Create a few nucbattlear-powered aircraft carrier battle groups. [BLEU: 59.7]

FT BETTER

Ground Truth: Ukraine does not supply or have plans to supply any armaments to the Government of South Sudan.
FT Prediction: Ukraine hadoes not supply orany have plans to supply any armsaments to the SGouvernmenth of South Sudan. [BLEU: 34.7]
SAFT Prediction: Ukraine hadoes not supply orany have planys to supply any armsaments to the SGouvernmenth of South Sudan. [BLEU: 12.6]

Ground Truth: The proposal may complicate the Bush administration's efforts to win an exemption for India to engage in nuclear trade.
FT Prediction: The proposedal maycould complicate the Bush administration's efforts to win Iandia exemption for India to engage in nuclear trade. [BLEU: 68.6]
SAFT Prediction: The proposedal maycould complicate the Buadminishtration administration's efforts to win Indian exemption forom India tfrom engage in nuclear trade. [BLEU: 29.7]

Ground Truth: In Virginia , it 's to benefit private business plans and not to serve the public interest .
FT Prediction: In Virginia ,it it 'hass ftor benefit privathe business plans ,and not to serve the public 'interest . [BLEU: 28.3]
SAFT Prediction: Itn Virginia ,it it 'iss ftor benefit privathe business plans ,and not to serve thepublic public 'interest . [BLEU: 8.4]

Ground Truth: 26/02/2010 14:32
FT Prediction: 26/02/2010 14:32 [BLEU: 31.6]
SAFT Prediction: 26February/02/2010 14:32 [BLEU: 15.0]

Ground Truth: Nepal (NP)
FT Prediction: Nepal (NP) [BLEU: 31.6]
SAFT Prediction: Nepal (NEP) [BLEU: 15.0]

BOTH GOOD

Ground Truth: proliferation; technology; international; politics
FT Prediction: proliferation; technology; international; politics [BLEU: 100.0]
SAFT Prediction: proliferation; technology; international; politics [BLEU: 100.0]

Ground Truth: They deserve it. They asked for that.
FT Prediction: They deserve it. They asked for that. [BLEU: 100.0]
SAFT Prediction: They deserve it. They asked for that. [BLEU: 100.0]

Ground Truth: Tell your ex that all communication needs to go through the lawyer.
FT Prediction: Tell your ex that all communication needs to go through athe lawyer. [BLEU: 82.7]
SAFT Prediction: Tell your ex that all communication needs to go through athe lawyer. [BLEU: 82.7]

Ground Truth: Xinhua News Agency , Rome , September 1st , by reporters Aiguo Yang and Changrui Huang
FT Prediction: Xinhua News Agency , Rome , September 1st , by reporters Aiguo Yang and Changrui Huang [BLEU: 81.5]
SAFT Prediction: Xinhua News Agency , Rome , September 1st , by reporters Aiguo Yang and Changrui Huang [BLEU: 81.5]

Ground Truth: International; Government; technology; politics; economy
FT Prediction: International; Government; technology; politics; economy [BLEU: 100.0]
SAFT Prediction: International; Government; technology; politics; economy [BLEU: 100.0]

BOTH BAD

Ground Truth: Haha
FT Prediction: Haha. [BLEU: 0.0]
SAFT Prediction: Haha. [BLEU: 0.0]

Ground Truth: Good Evening Digicel.
FT Prediction: Good eEvening,Digicel. [BLEU: 9.1]
SAFT Prediction: Good eEvening,Digicel,. [BLEU: 9.1]

Ground Truth: To help the survivors of the Gulf.
FT Prediction: ToHelp help those survivors inof the Gulf.region [BLEU: 3.7]
SAFT Prediction: ToSurv hsurvivelp those survivors inof the Gulf. [BLEU: 7.7]

Ground Truth: Please tell us our to pray because almost not believe in god and our prayer never arrived to god.
FT Prediction: Please tell us howur toprayers pray inwbecause wialmosthnout believeing in Ggod and heour prayers willnever arrived ato himgod. [BLEU: 6.2]
SAFT Prediction: Please,tell us howurto praybercauspray,we wealmostnot believeing in Ggod and our prayers willnever arrived to god. [BLEU: 3.4]

Ground Truth: You may think that ’s not rational land use .
FT Prediction: You cmany think that i’snothrational ltheand use use. [BLEU: 5.4]
SAFT Prediction: You cmany think that ’snotratiosusingal land usinguse use. [BLEU: 5.7]

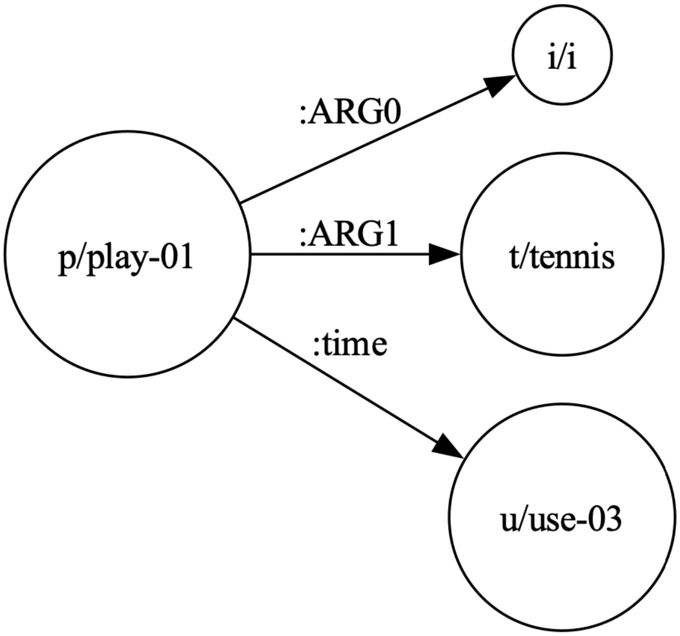
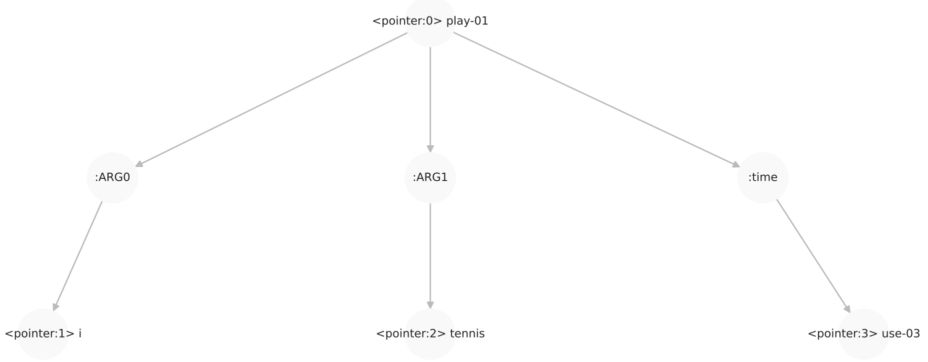


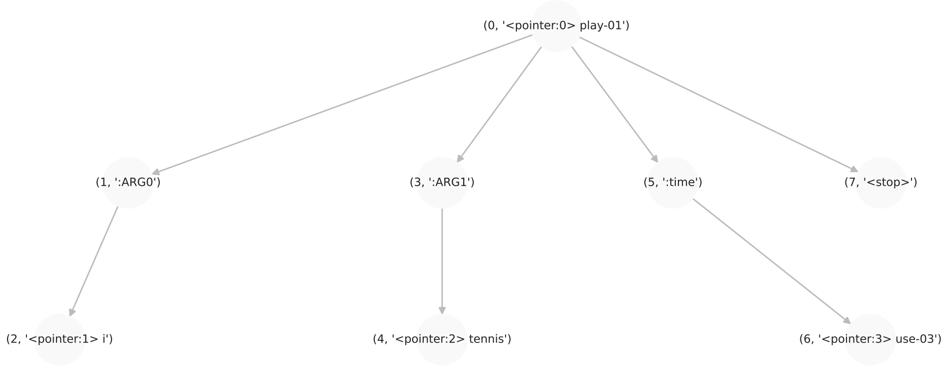
Figure 11: Original AMR graph for the sentence “I used to play tennis”.

```
<P0> play-01 :ARG0 <P1> i :ARG1 <P2> tennis :time <P3> use-03 <stop>
```

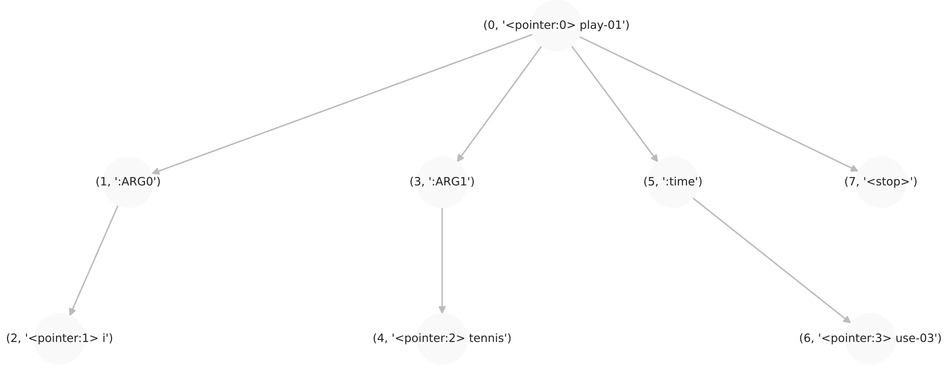
(a) BFS Linearization



(b) TOSUBGRAPH



(c) ROLEEXPAND, ADDSTOPNODES, and $\sigma_{\mathcal{A}}^{-1}$



(d) MERGE

Figure 12: Overview of preprocessing steps for the AMR corresponding to the sentence: “I used to play tennis”.

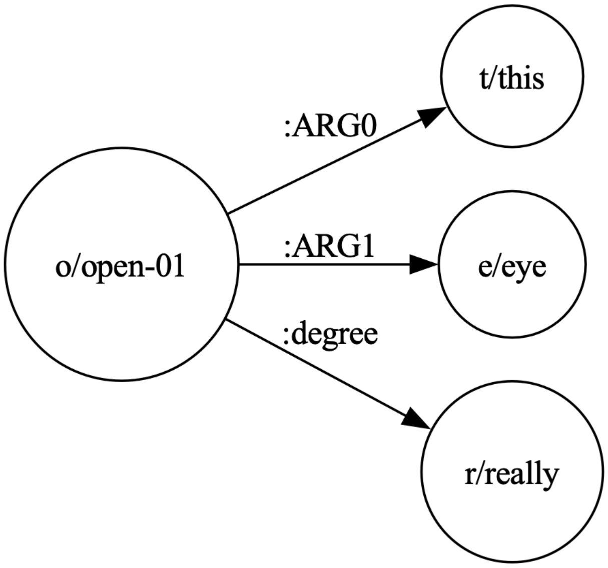
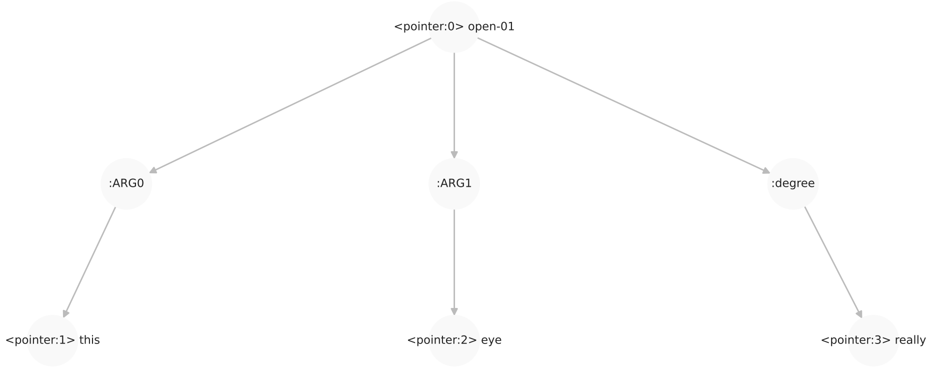


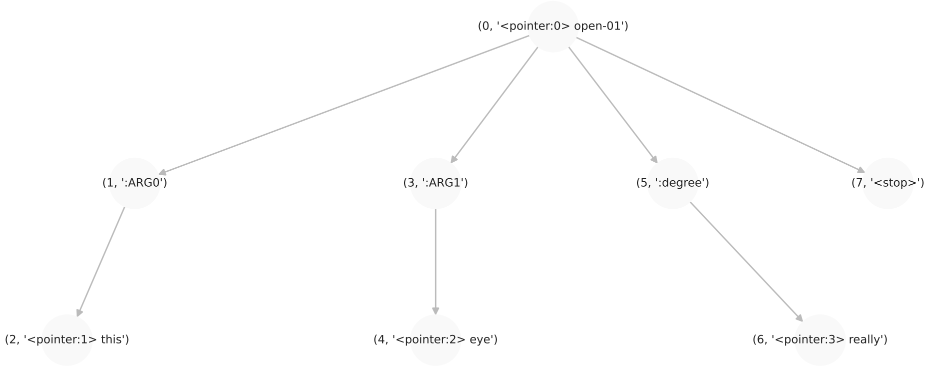
Figure 13: Original AMR graph for the sentence “This is really eye-opening”.

```
<P0> open-01 :ARG0 <P1> this :ARG1 <P2> eye :degree <P3> really <stop>
```

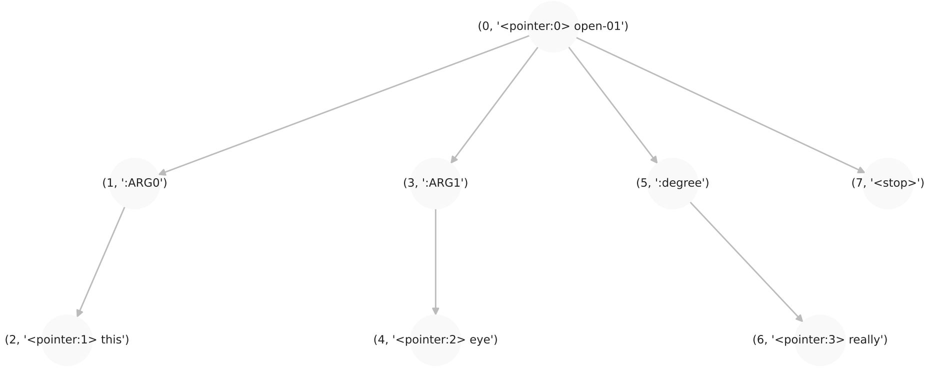
(a) BFS Linearization



(b) ToSubgraph



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



(d) Merge

Figure 14: Overview of preprocessing steps for the AMR corresponding to the sentence: “This is really eye-opening”.

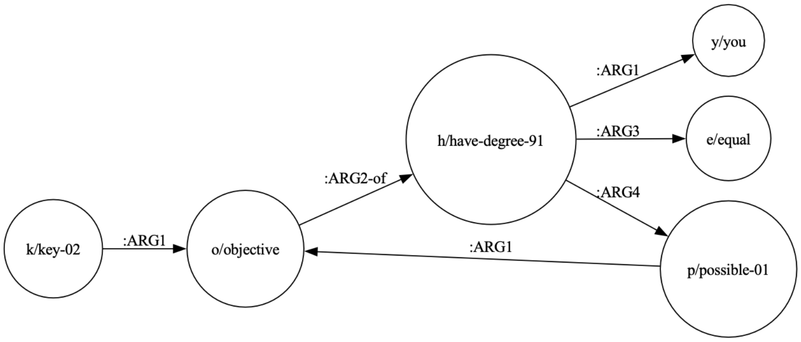
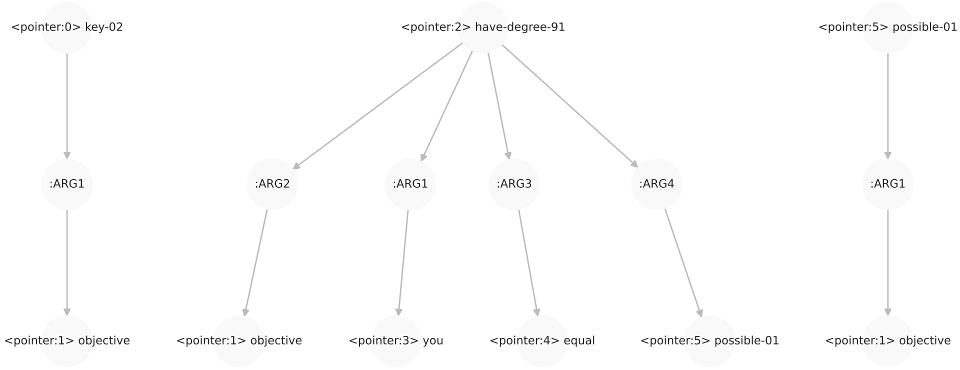


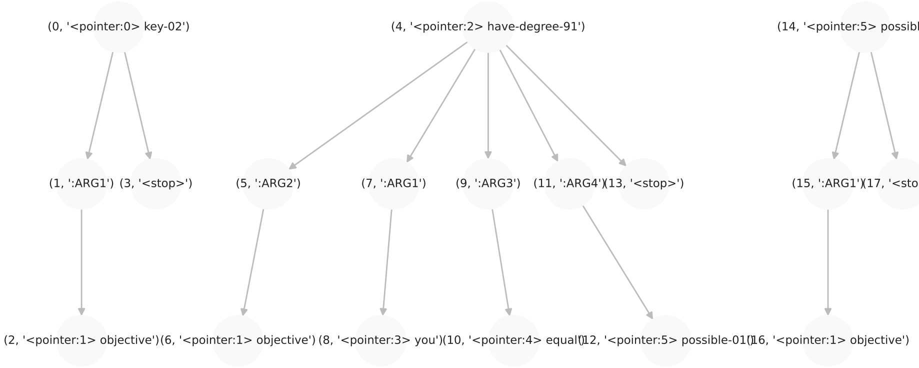
Figure 15: Original AMR graph for the sentence “The key is to be as objective as possible”.

```
<P0> key-02 :ARG1 <P1> objective <stop> <P2> have-degree-91 :ARG2 <P1>
:ARG1 <P3> you :ARG3 <P4> equal :ARG4 <P5> possible-01 <stop> <P5>
:ARG1 <P1> <stop>
```

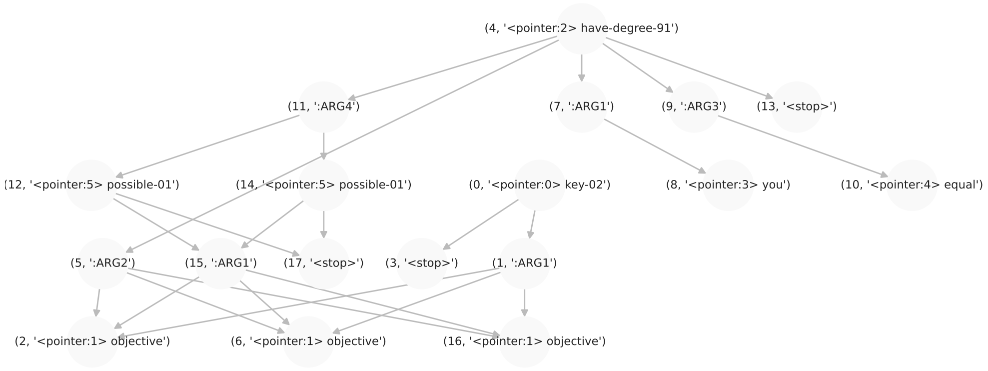
(a) BFS Linearization



(b) ToSubgraph



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



(d) Merge

Figure 16: Overview of preprocessing steps for the AMR corresponding to the sentence: “The key is to be as objective as possible”.

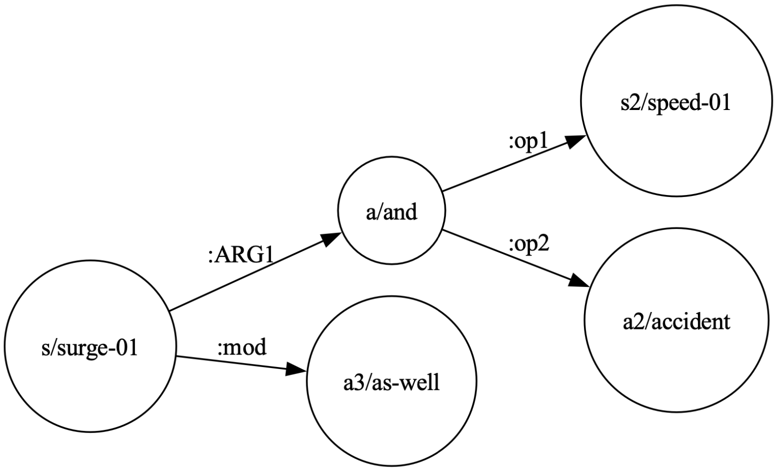
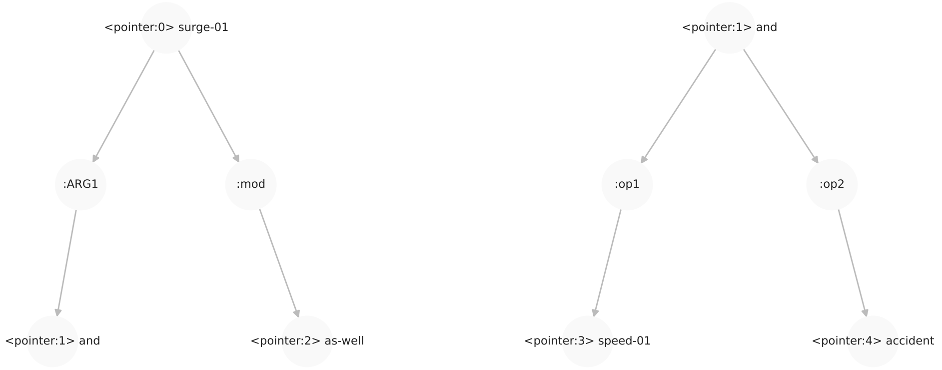


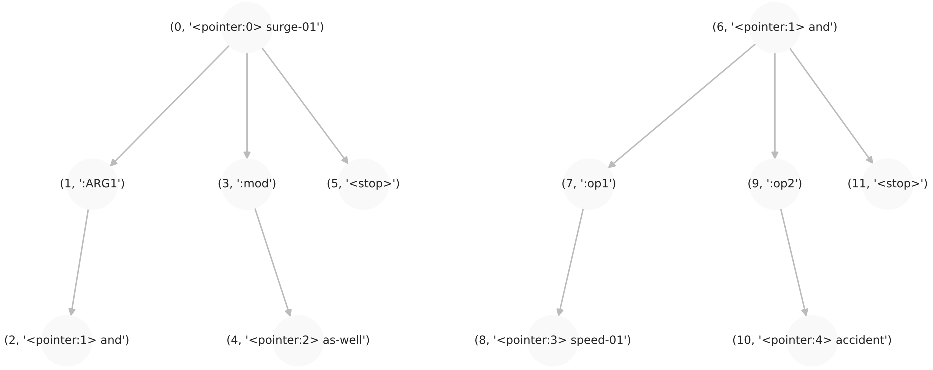
Figure 17: Original AMR graph for the sentence “Speeding and accidents have surged as well”.

```
<P0> surge-01 :ARG1 <P1> and :mod <P2> as-well <stop> <P1> :op1 <P3>
speed-01 :op2 <P4> accident <stop>
```

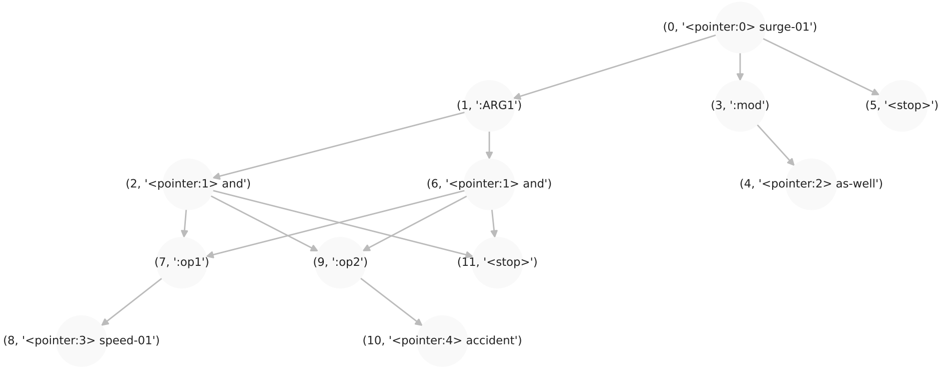
(a) BFS Linearization



(b) TOSUBGRAPH



(c) ROLEEXPAND, ADDSTOPNODES, and $\sigma_{\mathcal{A}}^{-1}$



(d) MERGE

Figure 18: Overview of preprocessing steps for the AMR corresponding to the sentence: "Speeding and accidents have surged as well".