Autoregressive Models for Knowledge Graph Generation

Anonymous authors
Paper under double-blind review

Abstract

Knowledge Graph (KG) generation requires models to learn complex semantic dependencies between triples while maintaining domain validity constraints. Unlike link prediction, which scores triples independently, generative models must capture interdependencies across entire subgraphs to produce semantically coherent structures. We present **ARK** (**A**uto-**R**egressive **K**nowledge Graph Generation), a family of autoregressive models that generate KGs by treating graphs as sequences of (head, relation, tail) triples. ARK learns implicit semantic constraints directly from data, including type consistency, temporal validity, and relational patterns, without explicit rule supervision. On the IntelliGraphs benchmark, our models achieve 89.2% to 100.0% semantic validity across diverse datasets while generating novel graphs not seen during training. We also introduce **SAIL**, a variational extension of ARK that enables controlled generation through learned latent representations, supporting both unconditional sampling and conditional completion from partial graphs. Our analysis reveals that model capacity (hidden dimensionality ≥ 64) is more critical than architectural depth for KG generation, with recurrent architectures achieving comparable validity to transformer-based alternatives while offering substantial computational efficiency. These results demonstrate that autoregressive models provide an effective framework for KG generation, with practical applications in knowledge base completion and query answering. Our code is available on `https://anonymous.4open.science/r/ARK-232F`.

1 Introduction

Knowledge Graphs (KGs) encode knowledge as graphs of entities connected by typed relations, powering applications from search engines to drug discovery (Hogan et al., 2021). However, even large-scale KGs such as Wikidata miss substantial world knowledge. Although Knowledge Graph Embedding (KGE) models address incompleteness (Bordes et al., 2013; Yang et al., 2015), they score each triple independently, failing to capture the interdependencies that define valid knowledge structures. This independence assumption becomes particularly problematic for complex facts requiring multiple related triples to represent accurately (Nathani et al., 2019). The KG generation task presents three key challenges that distinguish it from existing KG modeling approaches: (1) *Semantic constraint satisfaction*: generated triples must collectively satisfy domain rules (temporal consistency, type constraints) without explicit supervision, (2) *Structural coherence*: entities must form connected subgraphs with valid relational patterns, and (3) *Joint distribution modeling*: unlike KGE models, which score each triple independently, we model $p(G)$ over complete graphs.

Consider representing "*Barack Obama was the US President from 2009-2017*" in a KG; this requires multiple interdependent triples that must also satisfy temporal constraints (start year $\leq$ end year), and type consistencies (only persons can be presidents). Traditional link predictors cannot ensure that these constraints are satisfied collectively, leading to semantically invalid predictions that undermine downstream reasoning tasks (Thanapalasingam et al., 2023). This is particularly critical for N-ary relations that inherently cannot decompose into independent binary predictions (Wen et al., 2016). In contrast to link prediction, KG generation addresses these limitations by modeling joint distributions over sets of triples, enabling the sampling of complete graphs that satisfy semantic constraints across all their components simultaneously.

Generative models can learn these interdependencies by modeling entire (sub)graphs rather than individual links (Xie et al., 2022). Previous work on generative models in the KG domain has primarily focused on

generating triples from text (Saxena et al., 2022; Chen et al., 2020) or learning embeddings for downstream tasks (Xiao et al., 2016; He et al., 2015), rather than learning distributions over complete graph structures. To our knowledge, no prior work has demonstrated the ability to sample entire, semantically valid KGs from learned probabilistic models. This raises a fundamental question: *What is required to effectively model $p_\theta(G)$ for Knowledge Graphs?* We observe that KGs can be naturally represented as sequences of triples (head, relation, tail), suggesting that autoregressive sequence models may be well-suited for this task.

We introduce **ARK** (**A**uto-**R**egressive **K**nowledge Graph Generation), a family of autoregressive models that generate KGs by sequentially predicting triples. Our models learn semantic constraints, including type consistency, temporal validity, and relational patterns, directly from data without explicit rule supervision. On the IntelliGraphs benchmark, ARK achieves 89.2% to 100.0% semantic validity across diverse datasets. We further present **SAIL** (**S**equential **A**uto-Regress**I**ve Knowledge Graph Generation with **L**atents), a lightweight probabilistic extension of ARK that enables controllable generation from learned latent distributions.

We focus on subgraph generation, producing graphs of 3 to 212 triples that satisfy domain constraints. This scope aligns with practical applications, including knowledge base completion, query answering, and data augmentation, where generating valid local structures is the primary requirement. Our contributions are as follows:

1. We introduce ARK, an autoregressive approach to Knowledge Graph generation that learns implicit semantic constraints from data, achieving 89.2% to 100.0% validity on the IntelliGraphs benchmark without explicit rule supervision;
2. We present SAIL, a variational extension that enables controlled generation through learned latent representations, supporting both unconditional sampling and conditional completion from partial graphs;
3. We show that model capacity (hidden dimensionality ≥ 64) matters more than architectural depth, with single-layer GRUs matching deeper transformer performance while offering computational efficiency;
4. We release our models and code, establishing baselines for future work on KG generation. Our code is available at `https://anonymous.4open.science/r/ARK-232F`.

2 Preliminaries

Knowledge Graph Generation We consider the task of generating semantically valid Knowledge Graphs $G = (E, R, T)$ where E is a set of entities, R is a set of relations, and $T \subseteq E \times R \times E$ is a set of triples. Unlike link prediction, which focuses on individual triple classification, our goal is to generate a collection of triples (*i.e.* subgraphs) that satisfy domain-specific semantic constraints while capturing interdependencies. This subgraph inference task is particularly crucial for N-ary relations and more complex facts that cannot decompose into independent binary predictions (Thanapalasingam et al., 2023). For example, temporal constraints require that *start year* precede *end year* across multiple triples, while type constraints ensure that only valid entity-relation combinations appear together. Models must generate and validate entire structures collectively rather than scoring individual triples. This is distinct from link prediction or generation of triples from text, as the models need to assign probability to and sample entire sets of interdependent triples. We emphasize that we address subgraph generation (coherent collections of 3-212 triples), which is the practical use case for many real-world applications including KG completion and query answering.

Definition 2.1 (Knowledge Graph Generation)**.** Given a training set of Knowledge Graphs $\mathcal{D} = \{G_1, ..., G_n\}$, learn a generative model $p_\theta(G)$ that can sample new graphs $G' \sim p_\theta$ such that G' satisfies semantic validity constraints $\mathcal{S}$ while not appearing in $\mathcal{D}$. The task is neither link prediction (predicting individual triples) nor text-to-KG extraction (generating KGs from text), but learning to generate complete graph structures that satisfy implicit semantic constraints. We use this for: (1) unconditional sampling (generating diverse valid graphs), (2) conditional completion (completing partial graphs), and (3) learning the underlying distribution for compression.

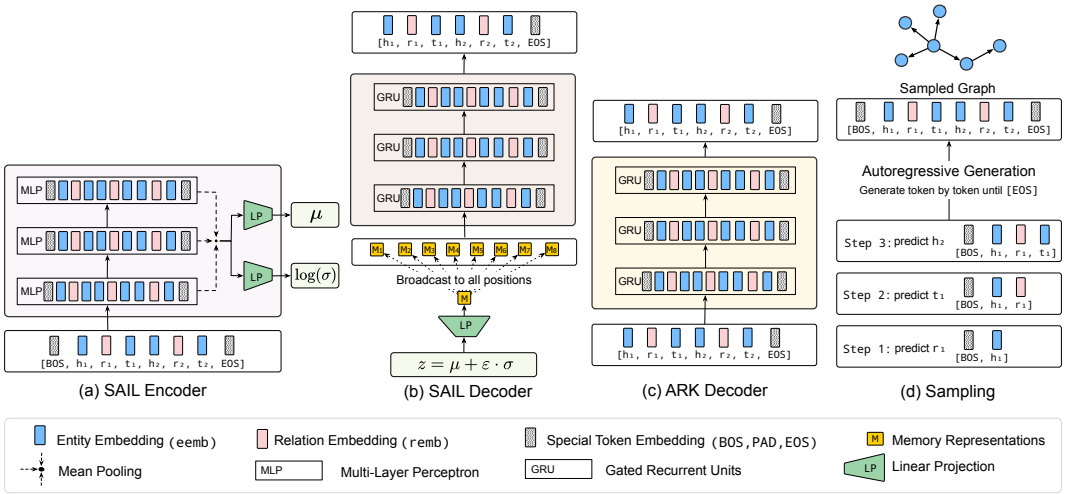


Figure 3.1: Overview of Model Architectures. **(a) SAIL Encoder:** Multi-layer perceptron (MLP) processes linearized KG sequences $[\texttt{BOS}, h_1, r_1, t_1, h_2, r_2, t_2, \ldots, \texttt{EOS}]$, with mean pooling to produce fixed-size representations. Linear projections generate latent distribution parameters μ and $\log \sigma$. **(b) SAIL Decoder:** GRU-based decoder conditions on sampled latent code $z \sim \mathcal{N}(\mu, \sigma^2)$ by broadcasting z to all sequence positions and concatenating with embeddings $[M_1, M_2, \ldots, M_n]$ at each timestep. **(c) ARK Decoder:** GRU decoder for ARK operates without latent conditioning, processing embedded sequences directly through stacked GRU layers. **(d) Sampling:** Autoregressive generation proceeds token-by-token with causal masking until EOS token or maximum length.

Definition 2.2 (Semantic Validity)**.** A graph G is semantically valid if it satisfies constraints $\mathcal{S} = \{s_1, ..., s_k\}$ where each s_i is a rule (e.g., type constraints, temporal consistency, relational dependencies). Concrete examples include: (1) start_year $\leq$ end_year (temporal consistency), (2) entity types must match relation requirements (e.g., only Person entities can have "birthplace" relations), (3) graphs must form valid paths (connected, acyclic, directional), (4) type consistency (directors are people, genres are categories), (5) similar type constraints plus connectivity. These constraints are not enforced during generation; the model must learn them from data.

Variational Inference To learn latent representations, we use the β-VAE framework (Kingma & Welling, 2013; Higgins et al., 2017), which aims to maximize the evidence lower bound (ELBO):

$$\mathcal{L}(\phi, \theta; G) = \mathbb{E}_{q_\phi(z|G)}[\log p_\theta(G|z)] - \beta \, \mathrm{KL}[q_\phi(z|G)||p(z)]. \tag{1}$$

3 Sequential Decoding for Knowledge Graph Generation

We present our approach to Knowledge Graph generation through sequential decoding. We first describe how graphs are linearized into token sequences, then introduce ARK (**A**uto-**R**egressive **K**nowledge Graph Generation), our autoregressive decoder model, followed by SAIL (**S**equential **A**uto-Regress**I**ve Knowledge Graph Generation with **L**atents), which extends ARK with a variational framework for controlled generation.

3.1 Graph Input Processing

To enable sequential generation, we linearize KGs into token sequences. A graph G containing triples $(h_1, r_1, t_1), ..., (h_n, r_n, t_n)$ is represented as $[\texttt{BOS}, h_1, r_1, t_1, h_2, r_2, t_2, ..., h_n, r_n, t_n, \texttt{EOS}]$, where BOS marks the sequence start and EOS indicates termination. These tokens provide explicit generation boundaries, enabling the model to learn proper initiation and termination conditions. We employ a unified vocabulary $\mathcal{V} = \{\texttt{BOS}, \texttt{PAD}, \texttt{EOS}\} \cup \mathcal{E} \cup \mathcal{R}$ that combines special tokens, entities, and relations into a single embedding space. Variable-length graphs are padded to a fixed maximum length $L_{\max}$ using PAD tokens for batched

training. During training, triple ordering within sequences is randomized to prevent the model from learning spurious positional patterns.[1]This randomization addresses permutation sensitivity: the model sees each graph in multiple orderings, learning order-invariant semantic constraints rather than positional patterns. Notably, our generated graphs are evaluated order-independently (checking whether the set of triples satisfies constraints), so generation quality does not depend on decoding order.

3.2 Autoregressive Knowledge generation (ARK)

ARK is an autoregressive model that generates KGs token-by-token, predicting each element conditioned on all previous tokens. We use Gated Recurrent Units (GRUs) (Cho et al., 2014) as the sequence model, exploiting the natural sequential structure of linearized graphs. See Appendix A.3.1 for architectural details.

The model is trained autoregressively with cross-entropy loss, conditioning on ground-truth previous tokens: $\mathcal{L}_{\text{ARK}} = -\sum_{t=1}^{T} \log p(x_t|x_{<t})$.

Generation During inference, ARK generates graphs sequentially starting from the `BOS` token. At each timestep t, the model computes the probability distribution $p(x_{t+1}|x_{\leq t})$ over the vocabulary. We select the next token through sampling controlled by temperature and top-k. Concretely, we divide logits by temperature T, keep only the top-k tokens, then retain the smallest prefix whose cumulative probability mass exceeds p (top-p), renormalize and sample one token. Decoding stops on `EOS` or when the maximum graph length has been reached. The generated sequence is parsed into triples by extracting consecutive (h, r, t) token triplets between `BOS` and `EOS` markers, with incomplete triples discarded during post-processing.

3.3 Sequential Autoregressive Knowledge Graph Generation with Latents (SAIL)

SAIL extends ARK by incorporating a variational autoencoder framework, similar to Bowman et al. (2016), enabling probabilistic generation from learned latent distributions, z. This extension allows for controlled generation and interpolation in latent space while maintaining the efficiency of GRU-based decoding. The VAE framework enables: (1) controlled generation through latent manipulation, (2) interpolation between graphs, and (3) conditional generation from partial graphs. The technical challenge is learning meaningful, continuous representations of discrete graph structures.

Encoder The encoder processes the input sequence through a multi-layer perceptron (MLP) to obtain a fixed-size representation.[2] Each input triple (h, r, t) is embedded as $[E_e[h]; E_r[r]; E_e[t]] \in \mathbb{R}^{3d}$, and we take a mean over the sequence to form a graph-level vector. The MLP consists of multiple dense layers, with the number of these layers chosen to match the number of stacked GRU layers in the decoder. The encoder then processes the aforementioned sequence through these layers, while ReLU is used as the activation function. The final hidden representation is projected to latent distribution parameters, $\boldsymbol{\mu}$ and $\log \boldsymbol{\sigma}^2$.

Latent Sampling We sample from the latent distribution using the reparameterization trick:

$$\mathbf{z} = \boldsymbol{\mu} + \boldsymbol{\sigma} \odot \boldsymbol{\epsilon}, \quad \boldsymbol{\epsilon} \sim \mathcal{N}(0, \mathbf{I}) \tag{2}$$

Following standard VAE practice, we use a fixed prior $p(z) = \mathcal{N}(0, I)$ rather than learning it, which acts as a regularizer. The learned components of our model are the encoder $q_\phi(\mathbf{z}|G)$, which maps graphs to latent distributions, and the decoder$p_\theta(G|\mathbf{z})$, which reconstructs graphs from latent codes. Despite using this simple fixed prior, our t-SNE visualizations in Figure 4 demonstrate that the learned posterior captures meaningful structure, with clear clustering by genre.

Decoder The decoder extends ARK's GRU architecture by conditioning on the latent variables, $\mathbf{z}$. The latent representation is first projected and used to initialize the decoder's hidden state: $\mathbf{h}_0 = \tanh(\mathbf{W}_{\text{init}}\mathbf{z} + \mathbf{b}_{\text{init}})$ To maintain global conditioning throughout generations, $\mathbf{z}$ is broadcast to all sequence positions. At each timestep, we concatenate the projected latent code with the input embedding: $\mathbf{x}'_t = [\mathbf{x}_t; \mathbf{W}_z\mathbf{z}]$ This

[1]This prevents leakage; *e.g.*, in the syn-paths dataset, which models KG paths, linearizing in path order makes the pattern trivially easy to learn.

[2]It may seem unusual to use an MLP here, which has no inductive bias for sequential data. In our experiments, a GRU-based encoder performed notably worse. We leave investigation of this counter-intuitive result to future work. Note that the t-SAIL architecture *does* provide a sequential inductive bias in both the encoder and decoder.

ensures that the global graph structure encoded in $\mathbf{z}$ influences every token prediction, allowing the decoder to maintain semantic consistency across the entire sequence. SAIL is trained by maximizing the ELBO (as shown in Equation 1).

Generation & Sampling To generate a graph using the model, we sample $\mathbf{z} \sim \mathcal{N}(0, \mathbf{I})$ from the prior distribution. We call this *unconditional generation*. Additionally, we define *conditional generation* where we encode a partial graph to obtain the posterior $q(\mathbf{z}|G_{\text{partial}})$, sample from it, and then complete the sequence. The generation then follows an autoregressive process where the probability of the complete graph factorizes as: $p_\theta(G|\mathbf{z}) = \prod_{t=1}^{T} p_\theta(x_t|x_{<t}, \mathbf{z})$. We use beam search with $\text{score}(x_{1:t}|\mathbf{z}) = \sum_{i=1}^{t} \log p_\theta(x_i|x_{<i}, \mathbf{z})$. Latent conditioning enables controlled generation by manipulating $\mathbf{z}$, we can interpolate between graphs or explore specific regions of the latent space to generate graphs with desired properties.

4 Evaluation

We evaluate a family of RNN and transformer-based models on the IntelliGraphs benchmark (Thanapalasingam et al., 2023), which consists of five datasets designed to test different aspects of Knowledge Graph generation.

Benchmark IntelliGraphs includes three synthetic datasets (syn-paths, syn-types, syn-tipr) with algorithmically verifiable semantics, ranging from simple path structures to temporal constraints requiring reasoning about time intervals, and two real-world Wikidata-derived datasets (wd-movies, wd-articles) capturing complex relational patterns from movie and academic publication domains. Synthetic datasets contain fixed-size graphs (3-5 triples) with small vocabularies (30-130 entities), while Wikidata datasets feature variable-size graphs (2-212 triples) with large entity vocabularies (24K-61K entities), providing diverse challenges for evaluating generation quality and semantic validity. Detailed dataset characteristics and semantic constraints are provided in Appendix A.2. To the best of our knowledge, IntelliGraphs is the only benchmark for KG generation, while other KG datasets focus on link prediction. While IntelliGraphs focuses on subgraph generation rather than complete KG generation, this aligns with practical applications where generating coherent subgraphs is the primary requirement, such as knowledge completion and query answering.

Baselines The probabilistic baselines from Thanapalasingam et al. (2023) decompose graph generation as $p(F) = p(S|E)p(E)$, where E represents entities and S represents structure. The *uniform* baseline samples from uniform distributions, providing estimates for compression bits by assuming equal likelihood for all configurations. The KGE-based baselines (TransE, ComplEx, DistMult) estimate $p(E)$ using entity frequencies with Laplace smoothing and $p(S|E)$ using learned scoring functions: TransE models relations as translations (Bordes et al., 2013), DistMult uses bilinear interactions (Yang et al., 2015), and ComplEx employs complex-valued embeddings (Trouillon et al., 2016). These models convert scores to probabilities through sigmoid functions and compute compression as $-\log_2 p(S|E) - \log_2 p(E)$. These baselines come directly from Thanapalasingam et al. (2023), which established them as generation baselines. While the KGE models weren't originally designed for generation, their failure (producing 76-100% empty graphs) demonstrates precisely why methods designed for independent triple scoring cannot handle joint graph generation. To study architectural choices, we also implement transformer-based variants: t-ARK uses a transformer decoder with causal self-attention, while t-SAIL extends this with a variational framework employing transformer encoders and decoders. These variants allow us to examine whether attention mechanisms provide benefits for KG generation. Other approaches like KGT5 (Kochsiek et al., 2023) and recent LLM-based methods address different tasks (text-to-KG extraction, question answering) rather than learning generative distributions over graph structures.

Evaluation Metrics We evaluate generation quality through three primary metrics: (1) *Semantic Validity* – the proportion of generated graphs that satisfy dataset-specific semantic constraints, measuring whether the model learns to respect domain rules without explicit supervision; (2) *Novelty* – the proportion of generated graphs not present in the training set, distinguishing genuine generation from memorization; and (3) *Compression* – the information-theoretic measure $-\log p(G)$ in bits, quantifying how efficiently the model encodes graph structure. For variational models, we additionally report the KL divergence between the approximate posterior and prior. These metrics collectively assess whether models capture the underlying data distribution while maintaining semantic coherence and generalization capability.

4.1 Compression Code Length

We express the negative log-likelihood, $-\log_2(p_\theta)$, in bits-per-graph. See Appendix A.3.2 for details. This measures both the ability to compress and to predict (Grünwald, 2007, Section 3.2).

Results Table 1 shows the compression performance across all models. ARK achieves strong compression rates across all datasets, with 27.65 bits for syn-paths (compared to 30.49 bits for the uniform baseline) and 23.48 bits for syn-tipr. On real-world datasets, ARK achieves the best overall compression with 98.19 bits for wd-movies and 205.24 bits for wd-articles, demonstrating efficient encoding of complex graph structures. While their compression on syn-types is higher (59.63 and 59.79 bits), both models compensate with strong semantic validity in generation tasks. By contrast, the variational models (SAIL and t-SAIL) report ELBO upper bounds rather than exact compression, as they use latent vectors $\mathbf{z}$ to capture graph structure. Their compression includes both reconstruction and KL divergence terms, with the KL component varying from nearly zero to syn-types (0.15 bits) to moderate values on other datasets (13-32 bits), indicating adaptive latent space usage on dataset complexity.

4.2 Sampling from Latent Variable, z

We assess the generative capabilities of SAIL through two complementary approaches: unconditional generation by sampling from the prior distribution $p_\theta(z)$, and conditional generation by providing partial graph sequences. These experiments test whether the learned latent space is well-structured and whether the model can generate semantically valid, novel graphs, demonstrating true generative modeling rather than mere memorization. For more details regarding the method and qualitative analysis, we refer the reader to Appendices A.3.3 and A.4, respectively.

Quantitative Results Table 1 shows unconditional graph generation results. ARK achieves high semantic validity across synthetic datasets: 99.95% on syn-paths, 100.00% on syn-tipr, and 89.22% on syn-types. SAIL demonstrates similarly strong performance with 92.50%, 98.45%, and 100.00% validity, respectively. Both models dramatically outperform KGE baselines (TransE, DistMult, ComplEx), which achieve less than 1% validity and produce 76–100% empty graphs, confirming that independent triple scoring cannot capture the joint structure required for KG generation. All generated graphs from our models are novel rather than memorizing training examples. For real-world datasets, ARK maintains 99.24% validity on wd-movies and 97.24% on wd-articles, while SAIL achieves 99.47% and 99.13% respectively, demonstrating robust performance despite increased complexity.

4.3 Interpolation in Latent Space

For SAIL and t-SAIL, we explore the structure of the learned latent space by interpolating between encoded representations of different graphs. This analysis reveals whether the model learns smooth, semantically meaningful transitions between graph structures, indicating a well-organized latent space where similar graphs cluster together and intermediate points correspond to valid hybrid structures. For more details regarding the method, we refer the reader to Appendix A.3.4.

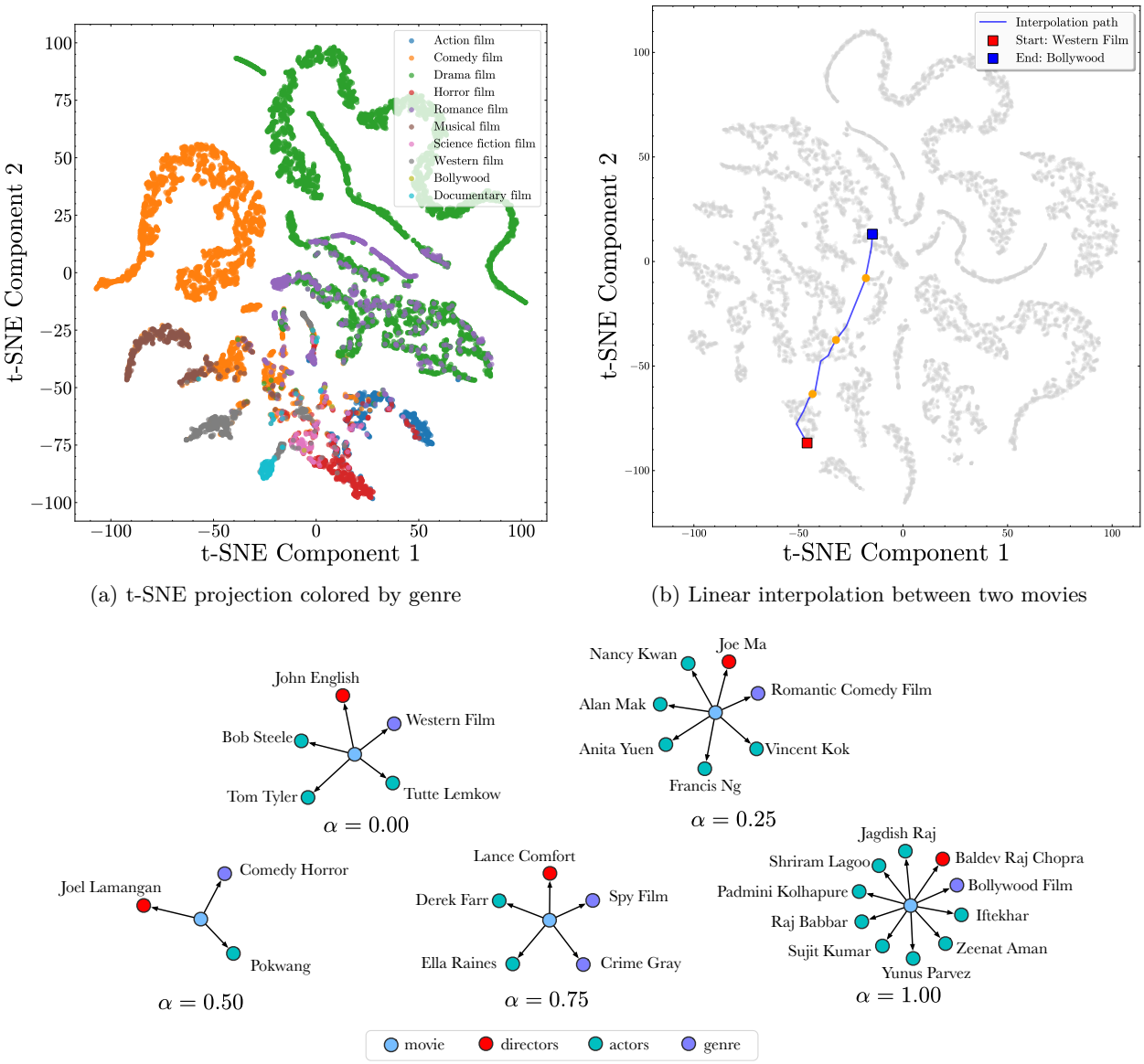
Quantitative Results The smoothness metrics reveal distinct patterns across dataset complexity. Comparing SAIL and t-SAIL (Figure 6 in Appendix), we observe that t-SAIL generally achieves better latent space organization. For syn-tipr, t-SAIL shows exceptional quality with near-perfect local smoothness (0.99) and global consistency (0.98), while SAIL achieves lower but still strong metrics (0.93 and 0.69, respectively). The architectural difference is most pronounced on syn-paths, where t-SAIL maintains moderate global consistency (0.36) compared to SAIL's much weaker performance (0.14), suggesting that transformer-based encoders better capture graph structure. Surprisingly, SAIL demonstrates superior performance on syn-types with high local smoothness (0.92) and global consistency (0.73), exceeding t-SAIL's metrics (0.82 and 0.60). The flip rates reveal interesting trade-offs: SAIL shows higher instability on most datasets (0.33 for syn-paths, 0.40 for wd-movies) compared to t-SAIL (0.20 and 0.15 respectively), though they achieve similar rates on syn-tipr (0.10 vs. 0.09). Real-world datasets (wd-movies) show both models achieving strong local smoothness (0.84 and 0.87) but with t-SAIL maintaining better global consistency (0.58 vs 0.49). These results suggest that while transformer encoders generally provide better latent space organization, the simpler

Datasets	Model	% Valid Graphs ↑	% Novel & Valid ↑	% Empty Graphs ↓	Compression Length (bits/graph) ↓
syn-paths	uniform	0	0	0	30.49
	TransE	0.25	0.25	76.55	49.89
	DistMult	0.69	0.69	85.41	54.39
	ComplEx	0.71	0.71	85.73	48.58
	t-SAIL	99.60	99.60	0	27.77
	SAIL	92.50	92.50	0	28.74
	t-ARK	97.39	97.39	0	**27.57**
	ARK	**99.95**	**99.95**	0	27.65
syn-tipr	uniform	0	0	0	61.61
	TransE	0	0	94.42	69.51
	DistMult	0	0	86.66	63.96
	ComplEx	0	0	96.05	67.51
	t-SAIL	100.00	100.00	0	26.30
	SAIL	98.45	98.45	0	27.14
	t-ARK	100.00	100.00	0	**23.34**
	ARK	**100.00**	**100.00**	0	23.48
syn-types	uniform	0	0	0	**36.02**
	TransE	0.21	0.21	84.56	48.26
	DistMult	0.13	0.13	87.53	47.69
	ComplEx	0.07	0.07	89.75	47.69
	t-SAIL	**100.00**	**100.00**	0	59.61
	SAIL	100.00	100.00	0	60.58
	t-ARK	87.07	87.07	0	59.79
	ARK	89.22	89.22	0	59.63
wd-movies	uniform	0	0	0	171.60
	TransE	0	0	85.39	208.60
	DistMult	0	0	87.07	202.68
	ComplEx	0	0	98.13	208.50
	t-SAIL	**99.83**	**99.83**	0	124.50
	SAIL	99.47	99.47	0	116.84
	t-ARK	98.33	98.33	0	114.49
	ARK	99.24	99.24	0	**98.19**
wd-articles	uniform	0	0	0	693.80
	TransE	0	0	95.42	910.65
	DistMult	0	0	100.00	887.30
	ComplEx	0	0	97.54	901.91
	t-SAIL	98.00	98.00	0	235.24
	SAIL	**99.13**	**99.13**	0	**199.55**
	t-ARK	95.37	95.37	0	224.25
	ARK	97.24	97.24	0	205.24

Table 1: Semantic validity and compression length in bits of the graphs generated. We sample graphs and check the novelty of the sampled graphs by comparing them against the training and validation sets. We use the test set for the calculation of the compression length when training has finished. The best performing models for each dataset are **bolded**. Baseline results are from the IntelliGraphs paper (Thanapalasingam et al., 2023). The full results are available in Tables 4 and 5 in the Appendix.

GRU-based SAIL can match or exceed transformer performance on certain structured datasets, particularly syn-types.

Qualitative Results Figure 4.1 demonstrates the learned latent space structure for the wd-movies dataset. The t-SNE projection (Figure 4.1 a) reveals distinct clustering by genre, indicating that SAIL learns to organize its latent space according to semantic film categories without explicit supervision. The linear interpolation experiment (Figure 4.1 b,c) traces a path between a Western film and a Romantic Comedy, with decoded graphs at intermediate points ($\alpha \in \{0, 0.25, 0.5, 0.75, 1\}$) demonstrating smooth transitions:



(a) t-SNE projection colored by genre

(b) Linear interpolation between two movies

(c) Decoded graphs at interpolation points $\alpha \in \{0.00, 0.25, 0.50, 0.75, 1.00\}$

Figure 4.1: Latent space visualization for the wd-movies dataset. (a) t-SNE projection shows clear clustering by genre. (b) Smooth interpolation paths connect different movie types. (c) Decoded graphs along the interpolation path show gradual transitions in cast and genre attributes, maintaining semantic validity throughout.

starting from a Western with actors Bob Steele and Tom Tyler, progressing through hybrid representations with mixed genre elements (Comedy Horror at $\alpha = 0.50$), and reaching a Thriller film with different cast members. While all intermediate graphs maintain a valid KG structure, the semantic coherence varies; intermediate points produce valid but potentially less realistic combinations of actors and genres, suggesting that semantic validity is preserved throughout interpolation, but semantic plausibility is highest near the training data manifold, consistent with typical VAE behavior on structured data.

4.4 Ablation Study

The ablation study in Appendix A.5 examines the relationship between model capacity and generation quality. We find that hidden dimensionality is more critical than network depth, with a clear performance threshold at $d_{\text{model}} = 64$. Single-layer GRU models with sufficient capacity (≥ 64 hidden units) match the validity of deeper architectures, suggesting that KG generation does not require complex multi-layer models. Compression efficiency improves with model complexity, yet lightweight architectures match transformer-based models in generation quality.

5 Related Work

While substantial progress has been made in related areas such as molecular graph generation and graph neural networks, the unique challenges of KG generation, including semantic consistency, relational diversity, and logical constraints, have only begun to be addressed. Recent advances in graph representation learning have paved the way for developing generative modeling of KGs.

Generative Modeling of KGs Cowen-Rivers et al. (2019) learn joint probability distributions over facts stored in Knowledge Graphs to estimate the predictive uncertainty of KGE models and evaluate their generative model using link prediction. TransG is a probabilistic model that learns the semantics of N-*ary* relations (Xiao et al., 2016). In contrast to this work, we focus on generating a collection of triples. Loconte *et al.* reinterprets the score functions of traditional KGEs as circuits, enabling efficient marginalization and sampling, thereby facilitating the generation of new triples consistent with existing KGs (Loconte et al., 2024). Notably, Galkin et al. (2024) proposes ULTRA, a foundation model for KG reasoning that achieves strong generalization across diverse KGs through a unified pre-training approach on multiple graphs, demonstrating that a single model can transfer reasoning capabilities across different knowledge domains without fine-tuning. Work focusing on triple completion from various sources includes: Zhang et al. (2024) which addresses triple set prediction, and Sun et al. (2018) which focuses on entity alignment. Neelakantan et al. (2015) uses RNNs for scoring paths in existing graphs (link prediction), whereas we generate complete novel graphs from learned distributions—a fundamentally different task.

Graph Transformers Machine learning on sets requires learning permutation-invariant functions (Zaheer et al., 2017). Various frameworks have been proposed that use attention mechanisms for graph representation learning (Kim et al., 2022; Yun et al., 2019; Zhuo et al., 2025; Zhao et al., 2025). Due to the fully-attentional nature of Transformers, they can be seen as a generalisation of Graph Neural Networks (Zaheer et al., 2017). In our work, we deal with a directed graph with labeled edges. Recent advances include GraphGPS (Rampášek et al., 2022), which combines message passing with global attention mechanisms, and NodeFormer (Wu et al., 2022), which efficiently computes all-pair interactions through kernelized softmax. Shirzad et al. (2023) introduce Exphormer, achieving linear complexity in graph transformers through virtual global nodes and expander graphs. Unlike these architectures that focus on encoding existing graphs, our work addresses the complementary problem of generating new KGs through autoregressive sequence modeling.

Graph Generative Models Deep graph generative models have predominantly focused on generating novel molecular structures, emphasizing chemical validity and stability (Li et al., 2018). Beyond molecular applications, models like GraphVAE and GraphRNN have been developed to capture complex graph structures through latent variable and autoregressive approaches. Kipf et al. (2020) introduced methods to infer symbolic abstractions from visual data and relational structures from observations. Recent developments include DiGress (Vignac et al., 2023), which applies discrete denoising diffusion to graph generation, and GraphARM (Kong et al., 2023), which combines autoregressive models with graph neural networks for scalable generation. Liu et al. (2024) propose GraphMaker, a diffusion-based approach that generates graphs by iteratively refining node features and edge structures. Our work extends these concepts by focusing on the semantic generation of KGs, learning implicit semantic constraints from background information without predefined rules.

Neuro-Symbolic Generative Models for KGs Combining distributed and symbolic representations, neuro-symbolic systems aim to combine the strengths of both paradigms (van Bekkum et al., 2021). Generative neuro-symbolic machine combines distributed and symbolic entity-based representations in a generative

latent variable model to infer object-centric symbolic representations from images Jiang & Ahn (2020). Balloccu et al. (2024) introduces KGGLM, a generative language model designed for generalizable KG representation learning in recommender systems, which exemplifies the integration of neural and symbolic approaches. A recent breakthrough comes from van Krieken et al. (2025), who introduce a neurosymbolic diffusion model that integrates logical constraints directly into the diffusion process. While these approaches explicitly incorporate symbolic reasoning into neural networks, our work demonstrates that autoregressive models can implicitly learn semantic constraints from data.

6 Conclusion

We have demonstrated that autoregressive sequence models provide an effective framework for Knowledge Graph generation. By representing KGs as sequences of (head, relation, tail) triples, our models learn to generate semantically valid graphs that satisfy domain constraints without explicit rule supervision. ARK and SAIL achieve 89.2% to 100.0% validity across diverse datasets, substantially outperforming KGE baselines that treat triples independently. The variational extension SAIL further enables controlled generation through learned latent representations, supporting both unconditional sampling and conditional completion from partial graphs. Our analysis reveals that hidden dimensionality matters more than architectural depth for this task, with single-layer GRU models matching the validity of deeper transformer architectures while offering computational efficiency. Our findings establish that autoregressive models can effectively learn the joint distribution over KG structures, implicitly capturing semantic rules. This opens avenues for KG completion, data augmentation, and query answering applications where generating valid local structures is the primary requirement.

Limitations Our work assumes a fixed vocabulary of entities and relations known at training time, limiting applicability to open-world scenarios where new entities emerge dynamically. While ARK and SAIL generate semantically valid graphs, in our experiments, we only test on relatively small Knowledge Graphs (3-212 triples). We focus on learning valid compositional patterns of known elements. While this limits applications requiring truly open-world generation, many practical use cases (KG completion, query answering over existing KGs) naturally operate under these assumptions. Moreover, our learned compositional patterns could potentially transfer to new entities through inductive embeddings in future work. IntelliGraphs focuses on subgraph generation rather than complete KG synthesis, which aligns with real-world applications that typically require coherent subgraphs. Additionally, the autoregressive formulation imposes a linear ordering on inherently unordered graph structures, though our experiments show that this does not significantly impact generation quality.

Future Work Several directions merit exploration: extending the ARK framework to handle *out-of-vocabulary* entities and relations through compositional embeddings or meta-learning approaches, investigating hierarchical generation strategies for larger graphs where local subgraphs are generated independently then composed, and making the learned semantic rules explicit rather than leaving them implicit in the model parameter would help to identify and mitigate learning undesired constraints that may stem from biases in the data. Since this work solves most challenges in the Intelligraphs benchmark, larger and more complex KG generation benchmarks are called for. Future directions include: (1) extending to open-world generation with compositional embeddings or meta-learning for handling new entities/relations dynamically, (2) scaling beyond 500 triples through hierarchical generation strategies (generating subgraphs then composing them), (3) integration with LLMs to combine our structured generation with natural language understanding, (4) making learned semantic rules explicit for better interpretability and bias detection, and (5) developing larger, more complex benchmarks since our models solve most IntelliGraphs challenges.

Ethics Statement Datasets on which our models are trained may contain societal biases and factual errors, which could propagate through the learning process and manifest in generated knowledge graphs. While our models achieve high semantic validity scores, they may still reproduce or amplify biases present in the training data, potentially generating graphs that reflect historical inequities or stereotypes. Additionally, the autoregressive generation process could produce factually incorrect but semantically valid triples, as the model learns logical rules rather than verifying the truth. We intend for ARK and SAIL to be treated as research prototypes to advance the field of KG generation, and should not be deployed in critical applications without thorough testing and safeguards. See Thanapalasingam et al. (2023) for a detailed analysis of the limitations of the datasets.

Reproducibility Statement We provide complete code and detailed configurations to ensure complete reproducibility of all experiments. Our implementation, including model architectures, training scripts, data preprocessing pipelines, and evaluation metrics, is available at `https://anonymous.4open.science/r/ARK-232F`. Experimental details, including hyperparameters, hardware specifications, and training procedures, are provided in Appendix A.1. We also release pre-trained model checkpoints for both ARK and SAIL to facilitate reproduction of our results and enable further research building upon our work. Detailed instructions for replicating each experiment, including expected runtimes and resource requirements, are provided in the repository.

References

Giacomo Balloccu, Ludovico Boratto, Gianni Fenu, Mirko Marras, and Alessandro Soccol. Kgglm: A generative language model for generalizable knowledge graph representation learning in recommendation. In *Proceedings of the 18th ACM Conference on Recommender Systems*, pp. 1079–1084, 2024.

Antoine Bordes, Nicolas Usunier, Alberto Garcia-Duran, Jason Weston, and Oksana Yakhnenko. Translating embeddings for modeling multi-relational data. In *Neural Information Processing Systems (NIPS)*, pp. 1–9, 2013.

Samuel Bowman, Luke Vilnis, Oriol Vinyals, Andrew Dai, Rafal Jozefowicz, and Samy Bengio. Generating sentences from a continuous space. In *Proceedings of the 20th SIGNLL conference on computational natural language learning*, pp. 10–21, 2016.

Wenhu Chen, Yu Su, Xifeng Yan, and William Yang Wang. KGPT: Knowledge-grounded pre-training for data-to-text generation. In Bonnie Webber, Trevor Cohn, Yulan He, and Yang Liu (eds.), *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pp. 8635–8648, Online, November 2020. Association for Computational Linguistics. doi: 10.18653/v1/2020.emnlp-main.697. URL `https://aclanthology.org/2020.emnlp-main.697/`.

Kyunghyun Cho, Bart Van Merriënboer, Dzmitry Bahdanau, and Yoshua Bengio. On the properties of neural machine translation: Encoder-decoder approaches. *arXiv preprint arXiv:1409.1259*, 2014.

Alexander I Cowen-Rivers, Pasquale Minervini, Tim Rocktaschel, Matko Bosnjak, Sebastian Riedel, and Jun Wang. Neural variational inference for estimating uncertainty in knowledge graph embeddings. *arXiv preprint arXiv:1906.04985*, 2019.

Mikhail Galkin, Xinyu Yuan, Hesham Mostafa, Jian Tang, and Zhaocheng Zhu. Towards foundation models for knowledge graph reasoning. In *International Conference on Learning Representations (ICLR)*, 2024.

Peter D Grünwald. *The minimum description length principle*. MIT press, 2007.

Shizhu He, Kang Liu, Guoliang Ji, and Jun Zhao. Learning to represent knowledge graphs with gaussian embedding. In *Proceedings of the 24th ACM international on conference on information and knowledge management*, pp. 623–632, 2015.

Irina Higgins, Loic Matthey, Arka Pal, Christopher Burgess, Xavier Glorot, Matthew Botvinick, Shakir Mohamed, and Alexander Lerchner. beta-vae: Learning basic visual concepts with a constrained variational framework. In *International conference on learning representations*, 2017.

Aidan Hogan, Eva Blomqvist, Michael Cochez, Claudia d'Amato, Gerard De Melo, Claudio Gutierrez, Sabrina Kirrane, José Emilio Labra Gayo, Roberto Navigli, Sebastian Neumaier, et al. Knowledge graphs. *ACM Computing Surveys (CSUR)*, 54(4):1–37, 2021.

Jindong Jiang and Sungjin Ahn. Generative neurosymbolic machines. *Advances in Neural Information Processing Systems*, 33:12572–12582, 2020.

Jinwoo Kim, Tien Dat Nguyen, Seonwoo Min, Sungjun Cho, Moontae Lee, Honglak Lee, and Seunghoon Hong. Pure transformers are powerful graph learners. In *Neural Information Processing Systems (NIPS)*, 2022.

Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *International conference on learning representations*, 2015.

Diederik P Kingma and Max Welling. Auto-encoding variational bayes. *International conference on learning representations*, 2013.

Thomas Kipf, Elise van der Pol, and Max Welling. Contrastive learning of structured world models. *International conference on learning representations*, 2020.

Adrian Kochsiek, Apoorv Saxena, Inderjeet Nair, and Rainer Gemulla. Friendly neighbors: Contextualized sequence-to-sequence link prediction. In Burcu Can, Maximilian Mozes, Samuel Cahyawijaya, Naomi Saphra, Nora Kassner, Shauli Ravfogel, Abhilasha Ravichander, Chen Zhao, Isabelle Augenstein, Anna Rogers, Kyunghyun Cho, Edward Grefenstette, and Lena Voita (eds.), *Proceedings of the 8th Workshop on Representation Learning for NLP (RepL4NLP 2023)*, pp. 131–138, Toronto, Canada, July 2023. Association for Computational Linguistics. doi: 10.18653/v1/2023.repl4nlp-1.11. URL `https://aclanthology.org/2023.repl4nlp-1.11/`.

Lingkai Kong, Jiachen Cui, Haotian Sun, Yuchen Zhuang, B. Aditya Prakash, and Chao Zhang. Autoregressive diffusion model for graph generation. In *International Conference on Machine Learning*, 2023.

Yujia Li, Oriol Vinyals, Chris Dyer, Razvan Pascanu, and Peter Battaglia. Learning deep generative models of graphs. *International Conference on Machine Learning*, 2018.

Mufei Liu, Eleonora Kreačić, Vamsi K. Potluru, and Pan Li. Graphmaker: Can diffusion models generate large attributed graphs? *Transactions on Machine Learning Research*, 2024.

Lorenzo Loconte, Nicola Di Mauro, Robert Peharz, and Antonio Vergari. How to turn your knowledge graph embeddings into generative models. *Advances in Neural Information Processing Systems*, 36, 2024.

Deepak Nathani, Jatin Chauhan, Charu Sharma, and Manohar Kaul. Learning attention-based embeddings for relation prediction in knowledge graphs. In Anna Korhonen, David Traum, and Lluís Màrquez (eds.), *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pp. 4710–4723, Florence, Italy, July 2019. Association for Computational Linguistics. doi: 10.18653/v1/P19-1466. URL `https://aclanthology.org/P19-1466/`.

Arvind Neelakantan, Benjamin Roth, and Andrew McCallum. Compositional vector space models for knowledge base completion. In Chengqing Zong and Michael Strube (eds.), *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, pp. 156–166, Beijing, China, July 2015. Association for Computational Linguistics. doi: 10.3115/v1/P15-1016. URL `https://aclanthology.org/P15-1016/`.

Ladislav Rampášek, Mikhail Galkin, Vijay Prakash Dwivedi, Anh Tuan Luu, Guy Wolf, and Dominique Beaini. Recipe for a general, powerful, scalable graph transformer. In *Advances in Neural Information Processing Systems*, volume 35, pp. 14501–14515, 2022.

Apoorv Saxena, Adrian Kochsiek, and Rainer Gemulla. Sequence-to-sequence knowledge graph completion and question answering. In *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 2814–2828, Dublin, Ireland, 2022. Association for Computational Linguistics. doi: 10.18653/v1/2022.acl-long.201. URL `https://aclanthology.org/2022.acl-long.201/`.

Hamed Shirzad, Ameya Velingker, Balaji Venkatachalam, Danica J Sutherland, and Ali Kemal Sinop. Exphormer: Sparse transformers for graphs. In *International Conference on Machine Learning*, pp. 31613–31632. PMLR, 2023.

Zequn Sun, Wei Hu, Qingheng Zhang, and Yuzhong Qu. Bootstrapping entity alignment with knowledge graph embedding. In *Proceedings of the Twenty-Seventh International Joint Conference on Artificial Intelligence, IJCAI-18*, pp. 4396–4402. International Joint Conferences on Artificial Intelligence Organization, July 2018. doi: 10.24963/ijcai.2018/611. URL https://doi.org/10.24963/ijcai.2018/611.

Thiviyan Thanapalasingam, Emile van Krieken, Peter Bloem, and Paul Groth. Intelligraphs: Datasets for benchmarking knowledge graph generation, 2023.

Théo Trouillon, Johannes Welbl, Sebastian Riedel, Éric Gaussier, and Guillaume Bouchard. Complex embeddings for simple link prediction. In *International conference on machine learning*, pp. 2071–2080. PMLR, 2016.

Michael van Bekkum, Maaike de Boer, Frank van Harmelen, André Meyer-Vitali, and Annette ten Teije. Modular design patterns for hybrid learning and reasoning systems: a taxonomy, patterns and use cases. *Applied Intelligence*, 51(9):6528–6546, 2021.

Emile van Krieken, Pasquale Minervini, Edoardo Maria Ponti, and Antonio Vergari. Neurosymbolic diffusion models. *Advances in neural information processing systems*, 2025.

Clement Vignac, Igor Krawczuk, Antoine Siraudin, Bohan Wang, Volkan Cevher, and Pascal Frossard. Digress: Discrete denoising diffusion for graph generation. In *International Conference on Learning Representations*, 2023.

Jianfeng Wen, Jianxin Li, Yongyi Mao, Shini Chen, and Richong Zhang. On the representation and embedding of knowledge bases beyond binary relations. *International Joint Conference on Artificial Intelligence*, 2016.

Qitian Wu, Wentao Zhao, Zenan Li, David Wipf, and Junchi Yan. Nodeformer: A scalable graph structure learning transformer for node classification. In *Advances in Neural Information Processing Systems*, 2022.

Han Xiao, Minlie Huang, and Xiaoyan Zhu. TransG : A generative model for knowledge graph embedding. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 2316–2325, Berlin, Germany, August 2016. Association for Computational Linguistics. doi: 10.18653/v1/P16-1219. URL https://aclanthology.org/P16-1219.

Xin Xie, Ningyu Zhang, Zhoubo Li, Shumin Deng, Hui Chen, Feiyu Xiong, Mosha Chen, and Huajun Chen. From discrimination to generation: Knowledge graph completion with generative transformer. In *Companion Proceedings of the Web Conference 2022*, pp. 162–165, 2022.

Bishan Yang, Wen-tau Yih, Xiaodong He, Jianfeng Gao, and Li Deng. Embedding entities and relations for learning and inference in knowledge bases. *International Conference on Learning Representations*, 2015.

Seongjun Yun, Minbyul Jeong, Raehyun Kim, Jaewoo Kang, and Hyunwoo J Kim. Graph transformer networks. *Advances in neural information processing systems*, 32, 2019.

Manzil Zaheer, Satwik Kottur, Siamak Ravanbakhsh, Barnabas Poczos, Russ R Salakhutdinov, and Alexander J Smola. Deep sets. *Advances in neural information processing systems*, 30, 2017.

Wen Zhang, Yajing Xu, Peng Ye, Zhiwei Huang, Zezhong Xu, Jiaoyan Chen, Jeff Z. Pan, and Huajun Chen. Start from zero: Triple set prediction for automatic knowledge graph completion. *IEEE Transactions on Knowledge and Data Engineering*, 36(11):7087–7101, November 2024. doi: 10.1109/TKDE.2024.3399832. URL https://arxiv.org/abs/2406.18166.

Qifang Zhao, Weidong Ren, Tianyu Li, Hong Liu, Xingsheng He, and Xiaoxiao Xu. Graphgpt: Generative pre-trained graph eulerian transformer. International Conference on Machine Learning, 2025.

Jiaming Zhuo, Yuwei Liu, Yintong Lu, Ziyi Ma, Kun Fu, Chuan Wang, Yuanfang Guo, Zhen Wang, Xiaochun Cao, and Liang Yang. Dualformer: Dual graph transformer. In *International Conference on Learning Representations*, 2025.

A Appendix

A.1 Experimental Details

We used the PyTorch library [3] to develop and test the models. All experiments were performed on a single-node machine with an Intel(R) Xeon(R) Gold 5118 (2.30GHz, 12 cores) CPU and 64GB of RAM, with a single NVIDIA A100 GPU (80GB of VRAM) or a single NVIDIA H100 GPU (80GB of VRAM). We used PyTorch's CUDA acceleration for model training and inference. We used the Adam optimizer with variable learning rates (Kingma & Ba, 2015). We monitored the training of the models using the Weights & Biases package [4]. All experiments use the same train/validation/test splits as the original IntelliGraphs benchmark (Thanapalasingam et al., 2023) to ensure fair comparison.

Hyperparameter Optimization For ARK and SAIL, the hyperparameters were automatically tuned using grid search {learning rate, batch size, number of epochs, latent dimension size [5], number of neurons and number of layers[6] } to get the best performance for the validation split. For reproducibility, we provide an extension description of the hyperparameters as YAML files under the *configs* directory on `https://anonymous.4open.science/r/ARK-232F`.

A.2 Dataset Details

The IntelliGraphs benchmark datasets test different aspects of semantic validity and structural complexity:

1. **syn-paths:** A synthetic dataset containing path graphs with simple semantics that can be algorithmically verified in linear time. These are acyclic graphs where edge directions follow the path structure.

2. **syn-types:** A synthetic dataset featuring typed entities and relations where type constraints on entities depend on the relation type, enforcing semantic consistency through type checking.

3. **syn-tipr:** A synthetic dataset containing subgraphs based on the *Time-indexed Person Role* (tipr) ontology pattern.[7] The semantics are defined by the tipr graph pattern, requiring temporal reasoning to generate valid time intervals.

4. **wd-movies:** Small knowledge graphs describing movies, extracted from Wikidata.[8] Each graph contains one existential node representing the movie, with entity nodes for director(s) connected via `has_director`, cast members connected via `has_actor`, and genres connected via `has_genre` relations.

5. **wd-articles:** Small knowledge graphs that describe research articles, extracted from Wikidata. Each graph contains one existential node representing the article, with entity nodes for author(s) connected via `has_author`, publication venues connected via `published_in`, and topics connected via `has_topic` relations.

[3] `https://pytorch.org/`
[4] `https://wandb.ai`
[5] for SAIL only
[6] Both encoder's and decoder's neurons and number of layers. For models without encoder the tuning for the number of layers and neurons was done the decoder part
[7] http://ontologydesignpatterns.org/wiki/Submissions:Time_indexed_person_role
[8] https://www.wikidata.org

Datasets	Dataset Size (Train/Val/Test)	Unique Entities	Relation Types	Triples per Graph
syn-paths	60,000/20,000/20,000	49	3	3
syn-types	60,000/20,000/20,000	30	3	3
syn-tipr	50,000/10,000/10,000	130	5	5
wd-movies	38,267/15,698/15,796	24,093	3	2-23
wd-articles	54,163/22,922/22,915	60,932	6	4-212

Table 2: Dataset characteristics for the IntelliGraphs benchmark. Synthetic datasets (syn-*) have fixed graph structures while Wikidata-derived datasets (wd-*) exhibit variable sizes. Entity counts represent unique entities across all graphs; edge counts indicate the number of triples per individual graph.

A.3 Methods

Here, we provide more details about the methods we used for the empirical analyses of ARK and SAIL.

A.3.1 Gated Recurrent Units (GRUs)

The ARK model employs a standard GRU decoder with hidden state $\mathbf{h}_t \in \mathbb{R}^d$ that evolves as:

$$\mathbf{r}_t = \sigma(\mathbf{W}_r \mathbf{x}_t + \mathbf{U}_r \mathbf{h}_{t-1} + \mathbf{b}_r) \tag{3}$$

$$\mathbf{z}_t = \sigma(\mathbf{W}_z \mathbf{x}_t + \mathbf{U}_z \mathbf{h}_{t-1} + \mathbf{b}_z) \tag{4}$$

$$\tilde{\mathbf{h}}_t = \tanh(\mathbf{W}_h \mathbf{x}_t + \mathbf{U}_h (\mathbf{r}_t \odot \mathbf{h}_{t-1}) + \mathbf{b}_h) \tag{5}$$

$$\mathbf{h}_t = (1 - \mathbf{z}_t) \odot \mathbf{h}_{t-1} + \mathbf{z}_t \odot \tilde{\mathbf{h}}_t \tag{6}$$

where $\mathbf{r}_t$ and $\mathbf{z}_t$ are reset and update gates respectively, $\mathbf{x}_t$ is the embedding of the current input token, and $\odot$ denotes element-wise multiplication. At each timestep, the hidden state is projected to vocabulary logits: $p(x_{t+1}|x_{\leq t}) = \text{softmax}(\mathbf{W}_o \mathbf{h}_t + \mathbf{b}_o)$.

A.3.2 Compression Length

For both ARK and SAIL, we compute the compression length to generate graphs as sequences. Since ARK is a decoder-only autoregressive model, we compute:

$$\text{Compression Length of } G = -\log_2(p_\theta(G)) = -\sum_{t=1}^{T} \log_2(p_\theta(x_t|x_{<t})) \tag{7}$$

where x_t represents the t-th token in the linearized graph sequence $[\texttt{BOS}, h_1, r_1, t_1, ..., \texttt{EOS}]$ and T is the sequence length. Each term represents the bits needed to encode the next token given the previous context.

For SAIL, the variational framework adds a latent variable z, resulting in an upper bound on compression length through the ELBO:

$$\text{Compression Length of } G \leq -\log_2(p(G|z)) + D_{\text{KL}}(q(z \mid G) \parallel p(z)) \tag{8}$$

$$= -\sum_{t=1}^{T} \log_2(p_\theta(x_t|x_{<t}, z)) + D_{\text{KL}} \tag{9}$$

The KL divergence term is computed as follows:

$$D_{\text{KL}}(q(z \mid G) \parallel p(z)) = \frac{1}{2} \sum_{i=1}^{d} \left(\mu_i^2 + \sigma_i^2 - 1 - \log(\sigma_i^2)\right) \cdot \log_2(e) \tag{10}$$

where d is the latent dimensionality and the factor $\log_2(e)$ converts from nats to bits. The autoregressive formulation naturally handles variable-length graphs through the sequential factorization, eliminating the need for separate structure and entity terms.

This provides an upper bound on the true compression length; the VAE's ELBO is a lower bound on log-likelihood, which, when negated, becomes an upper bound on compression. The bound is particularly relevant as the autoregressive decoder must account for uncertainty in token ordering during generation.

A.3.3 Sampling from Latent Variable, z

We conduct two types of generation experiments:

1. *Unconditional Generation:* We sample 10,000 random latent codes from the standard normal prior distribution $p(z) = \mathcal{N}(0, I)$ and decode them into complete graphs using beam search with beam width $k = 3$. Each decoded graph is analyzed for: (1) semantic validity according to dataset-specific constraints, (2) novelty by checking against the training and validation sets, and (3) non-emptiness to ensure the model generates meaningful structures rather than null graphs.

2. *Conditional Generation:* We evaluate the model's ability to complete partial graphs by providing incomplete sequences as prompts. For each test graph, we provide the first n tokens (*e.g.*, $[\texttt{BOS}, h_1, r_1, t_1]$) and generate the remaining sequence autoregressively. We vary the conditioning length and measure: (1) the semantic validity of the completed graph and (2) the diversity of completions when sampling with different random seeds.

A.3.4 Interpolation in Latent Space

We conduct both quantitative and qualitative analyses of the latent space structure:

1. *Quantitative Analysis:* We measure latent space smoothness through four metrics: (1) *Local Smoothness* – average Jaccard similarity between consecutive decoded graphs along random walks in latent space with step size $\epsilon = 0.1$, measuring whether small movements produce similar graphs; (2) *Global Consistency* – Jaccard similarity between each step and the anchor point, measuring drift from the starting graph; (3) *Flip Rate* – fraction of steps that produce different decoded graphs, with lower rates indicating larger basins of attraction in latent space; and (4) *Average Basin Length* – mean number of consecutive interpolation steps that decode to identical graphs, quantifying the granularity of the learned representation. For each metric, we sample multiple anchor points and random directions, taking 10-30 steps along each trajectory.

2. *Qualitative Analysis:* We visualize the latent space structure using two approaches: (1) *2D Projection* – we encode all test graphs and project their latent representations to 2D using t-SNE, coloring points by semantic attributes (genre for wd-movies) to observe clustering patterns; and (2) *Linear Interpolation* – we select pairs of semantically distinct graphs, encode them to obtain z_1 and z_2, then decode intermediate points $z_\alpha = (1 - \alpha)z_1 + \alpha z_2$ for $\alpha \in [0, 1]$ at regular intervals to examine the semantic coherence of interpolated graphs.

A.4 Qualitative Analysis of Conditional Sampling

Qualitative Results We test whether SAIL has learned meaningful latent representations that capture director-specific collaborative patterns and genre preferences, despite never being explicitly trained on individual directorial styles. Figure A.1 shows representative examples of conditional generation for director-specific movie graphs. When conditioned on "Tim Burton" as the director, the model successfully generates graphs featuring his frequent collaborators (Helena Bonham Carter, Christopher Lee) and characteristic genres (Comedy Film, Musical Film). SAIL captures Burton's tendency to work repeatedly with the same ensemble cast, demonstrating learned patterns of directorial collaboration. In contrast, the Wes Anderson generation fails to capture his distinctive style. This disparity in generation quality likely reflects differences in dataset representation; Burton's more frequent appearances and consistent casting patterns in the training

Figure A.1: Graphs generated by ARK conditioned on director entities for Wes Anderson **(left)** and (b) Tim Burton **(right)**. Node colors indicate entity types: movie (blue), directors (red), actors (green), and genres (purple).

data enabled better pattern learning, while Anderson's style may have been underrepresented. Despite these variations in director-specific accuracy, both generated graphs maintain semantic validity as movie KGs, indicating that the SAIL has learned general graph structure.

A.5 Ablation Study

We systematically analyze the contribution of key architectural components through two ablation experiments on the syn-paths dataset, examining both model capacity and architectural choices.

Method We conduct two complementary ablation studies:

1. *Architectural Hyperparameter Analysis:* We vary the number of GRU layers $n_{\text{layers}} \in \{1, 2, 3, 4, 5\}$ and model dimensions $d_{\text{model}} \in \{2, 4, 8, 16, 32, 64, 128, 256, 512\}$ while keeping other hyperparameters fixed. For each configuration, we train the model until convergence and evaluate generation by measuring the percentage of semantically valid and novel graphs. We also test the relative importance of network depth versus hidden dimensionality on generation quality.

2. *Architecture Ablation:* We systematically replace transformer components with simpler architectures to assess their contribution: (1) *MLP Encoder* – replaces the transformer encoder with a multi-layer perceptron while preserving positional encoding; (2) *GRU Decoder* – replaces the transformer decoder with a GRU-based sequential decoder; and (3) *MLP Encoder & GRU Decoder* – combines both modifications, using an MLP encoder and GRU decoder. Each variant maintains comparable parameter counts to the transformer baseline for fair comparison.

Architectural Hyperparameter Analysis Results In Figure A.2, the model dimension has a substantially stronger impact on generation quality than network depth. Varying the number of layers from 1 to 5 produces relatively stable performance around 45% valid & novel rate, though with high variance across configurations. In contrast, the center panel demonstrates a sharp performance threshold: models with fewer than 16 hidden units achieve near-zero validity rates, while those with $d_{\text{model}} \geq 64$ consistently achieve 70-95% validity. The right panel's scatter plot confirms this pattern across individual runs, showing clear stratification by model dimension rather than layer count (indicated by color). These findings suggest that for KG generation on syn-paths dataset, a single-layer GRU with sufficient hidden units ($\geq$64) can match or exceed the performance of deeper networks, supporting our claim that architectural simplicity does not compromise generation quality when coupled with appropriate capacity.

Architecture Ablation Results To better understand the contribution of architectural choices, we compare our full transformer-based model t-SAIL against simplified variants: SAIL, which replaces the transformer encoder and decoder with an MLP encoder and a GRU decoder, an MLP encoder (paired with a transformer decoder), t-ARK, a decoder-only transformer model, and ARK, a GRU decoder-only model.

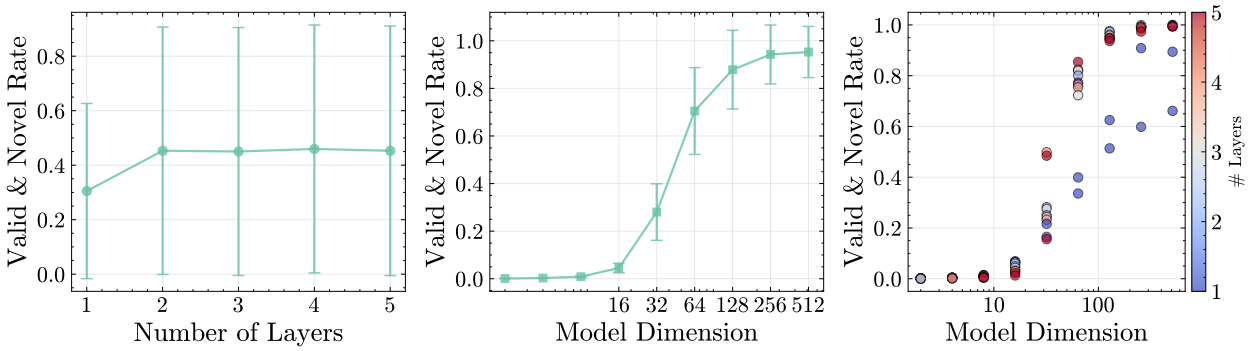


Figure A.2: Effect of architectural hyperparameters on the semantic validity and novelty. **(Left)** Valid & Novel rate as a function of the number of GRU layers, showing stable performance across depths with high variance. **(Center)** Performance variation with model dimension (hidden units), demonstrating a sharp improvement threshold around 64 dimensions, followed by consistent high performance. **(Right)** Scatter plot of individual experimental runs showing the relationship between model dimension and generation quality, with color indicating the number of layers.

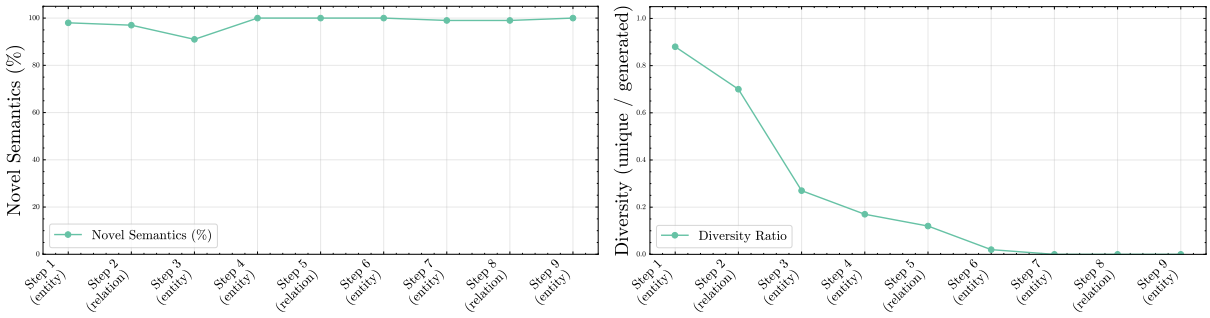
These ablations allow us to isolate the effect of transformer components in both the encoder and the decoder, and to assess whether an encoder is required for KG generation at all. In addition to generation quality and compression efficiency, we also report relative training time, as computational efficiency is often a limiting factor in scaling generative models. Table 3 demonstrates that transformer components, while improving generation quality, are not strictly necessary for effective knowledge graph modeling. Sequential decoders are consistently the most efficient: ARK trains at **0.09**–**0.27**,× the baseline time (i.e., **3.7**–**11**× faster) with near baseline validity across datasets, and its sequential inductive bias is competitive for decoding *e.g.*, syn-tipr (23.48 bits, on par with *t*-ARK 's 23.34) and wd-movies (**98.19** bits, best overall). Meanwhile, SAIL yields the best compression on wd-articles (**199.55** bits), indicating that modest latent structure plus a GRU decoder can improve efficiency on complex, real-world graphs. Taken together, these results suggest that, for KG generation, a strong sequential decoder often dominates architectural choice, and the extra cost of full transformers, especially in the decoder, may be hard to justify when compute is constrained.

A.6 Conditioned Generation

Figure A.3 shows that conditioned generation is also possible for the ARK model, which allows the model to generate KGs and simultaneously enforces specific constraints. Entities or relations are fixed in place in the positions of interest, and then we decode the remaining tokens with constrained sampling (temperature/top-k/top-p). Figure A.3a shows that the novelty and validity of the generated structures remain high for all steps of the conditioning process, an indication that the model can produce triples and, consequently, graphs that are semantically correct. At the same time, as seen in Figure A.3b, the diversity of the generated graphs drops dynamically as more entities and relations are added. This makes sense as the population of probable samples narrows with each additional constraint and limits the generative freedom of the model.

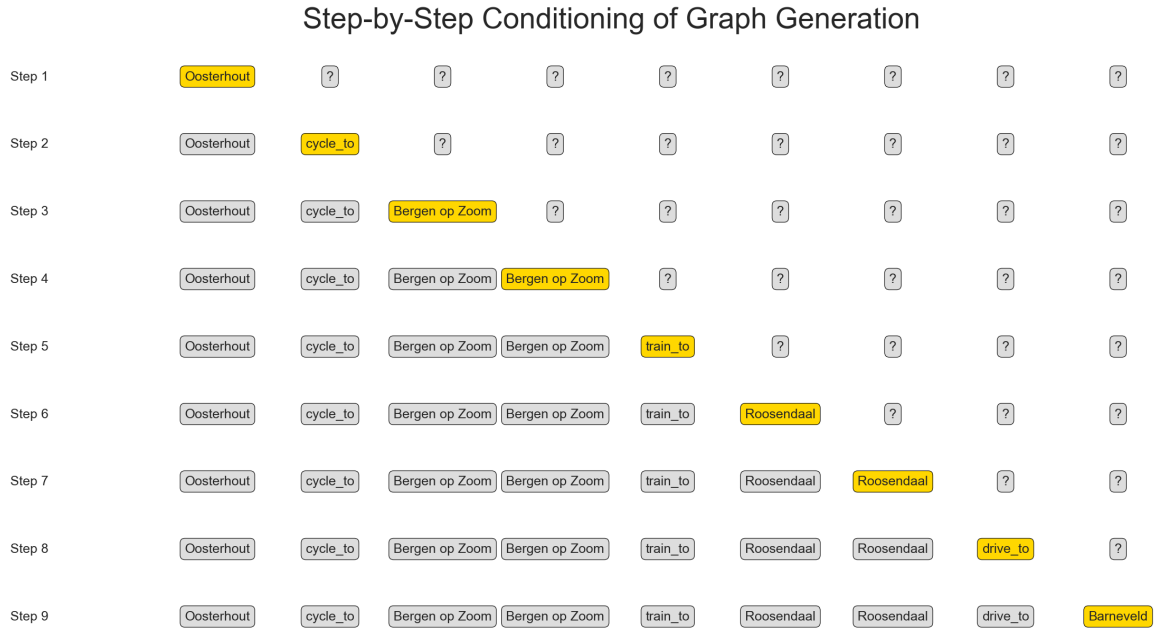
A.7 Additional Comments about ARK & SAIL

Variable Graph Length It is desirable to learn latent graph structures of varying sizes. In natural language processing, language models utilize special tokens to indicate the end of a sequence. Following a similar approach, we model variable-length KGs by linearizing graphs into a sequence of tokens and introducing boundary tokens. We always introduce `BOS` as the initial token and terminate generation upon emitting `EOS`, while using `PAD` for mini-batching. This simple setting allows the decoder to learn *when* to stop and *how large* the generated graphs should be, ensuring that the length distribution is learned. During inference, beam search halts on `EOS`, leading to the production of graphs of different sizes without any post hoc trim. In order to avoid length bias, we randomize triple order during training. In the probabilistic



(a) Stepwise conditioning for novelty and validity.

(b) Effect of conditioning on diversity.



(c) Example of syn-paths conditioned generation

Figure A.3: Effect of progressive conditioning on Knowledge Graph generation for the syn-paths dataset. Subfigure (a) quantifies novelty and validity under increasing conditioning, (b) shows the corresponding reduction in sample diversity, and (c) provides an example of a conditioned generation where the model completes a partially specified graph.

Datasets	Model	% Valid Generation ↑	% Novel Graphs ↑	Compression (bits) ↓	Training Time ↓
syn-paths	*t*-SAIL	99.60	100.00	27.77	1.00
	SAIL	92.50	100.00	28.74	0.21
	MLP Encoder	99.80	100.00	27.35	0.55
	t-ARK	97.39	100.00	27.57	0.12
	ARK	99.95	100.00	27.65	0.09
syn-tipr	*t*-SAIL	100.00	100.00	26.30	1.00
	SAIL	98.45	100.00	27.14	0.17
	MLP Encoder	99.48	100.00	26.30	0.20
	t-ARK	100.00	100.00	23.34	0.17
	ARK	100	100.00	23.48	0.09
syn-types	*t*-SAIL	100.00	100.00	59.61	1.00
	SAIL	100.00	100.00	60.58	0.39
	MLP Encoder	93.27	100.00	59.33	0.41
	t-ARK	87.07	100.00	59.79	0.18
	ARK	89.22	100.00	59.63	0.09
wd-movies	*t*-SAIL	99.83	100.00	124.50	1.00
	SAIL	99.47	100.00	116.84	0.24
	MLP Encoder	99.44	100.00	118.64	0.36
	t-ARK	98.33	100.00	114.49	0.23
	ARK	99.24	100.00	98.19	0.21
wd-articles	*t*-SAIL	98.00	96.00	235.24	1.00
	SAIL	99.13	100.00	199.55	0.42
	MLP Encoder	97.7	100.00	206.23	0.48
	t-ARK	95.37	100.00	224.25	0.33
	ARK	97.24	100.00	205.24	0.27

Table 3: Architectural ablation study comparing ARK against simplified architectures with MLP encoders and GRU decoders. We evaluate model variants across five datasets using generation quality metrics (percentage of valid and novel graphs), compression efficiency (bits required for latent representation), and computational efficiency (training time relative to *t*-SAIL baseline).

variant (SAIL), the latent $\mathbf{z}$ conditions the entire sequence and this yields consistent length control across all samples, while at the same time preserving variability.

B Additional Tables

Datasets	Model	% Valid Graphs ↑	% Novel & Valid ↑	% Novel Graphs ↑	% Empty Graphs ↓
syn-paths	uniform	0	0	100.00	0
	TransE	0.25	0.25	23.45	76.55
	DistMult	0.69	0.69	14.59	85.41
	ComplEx	0.71	0.71	14.27	85.73
	t-SAIL	99.60	99.60	100.00	0
	SAIL	92.50	92.50	100.00	0
	t-ARK	97.39	97.39	100.00	0
	ARK	**99.95**	**99.95**	100.00	0
syn-tipr	uniform	0	0	100.00	0
	TransE	0	0	5.58	94.42
	DistMult	0	0	13.34	86.66
	ComplEx	0	0	4.95	96.05
	t-SAIL	100.00	100.00	100.00	0
	SAIL	98.45	98.45	100.00	0
	t-ARK	100.00	100.00	100.00	0
	ARK	**100.00**	**100.00**	100.00	0
syn-types	uniform	0	0	100.00	0
	TransE	0.21	0.21	15.44	84.56
	DistMult	0.13	0.13	12.46	87.53
	ComplEx	0.07	0.07	10.25	89.75
	t-SAIL	100.00	**100.00**	**100.00**	0
	SAIL	100.00	100.00	100.00	0
	t-ARK	87.07	87.07	100.00	0
	ARK	89.22	89.22	100.00	0
wd-movies	uniform	0	0	100.00	0
	TransE	0	0	14.61	85.39
	DistMult	0	0	12.93	87.07
	ComplEx	0	0	1.87	98.13
	t-SAIL	**99.83**	**99.9**	**100**	0
	SAIL	99.47	99.47	100.00	0
	t-ARK	98.33	98.33	100.00	0
	ARK	99.24	99.24	100.00	0
wd-articles	uniform	0	0	100.00	0
	TransE	0	0	4.58	95.42
	DistMult	0	0	0	100.00
	ComplEx	0	0	2.46	97.54
	t-SAIL	98.00	98.00	100.00	0
	SAIL	**99.13**	**99.13**	100.00	0
	t-ARK	95.37	95.37	100.00	0
	ARK	97.24	97.24	99.99	0

Table 4: Semantic validity of the graphs generated. We sample graphs and check the novelty of the sampled graphs by comparing them against the training and validation sets. The best performing models for each dataset are **bolded**. Baseline results are from the IntelliGraphs paper (Thanapalasingam et al., 2023).

Datasets	Models	Compression Length (bits)			
		G	S	E	D_{KL}
syn-paths	uniform	30.49	12.80	17.69	-
	TransE	49.89	16.19	33.69	-
	ComplEx	54.39	20.71	33.69	-
	DistMult	48.58	14.90	33.69	-
	t-SAIL	27.77	-	14.47	13.30
	SAIL	28.74	-	18.41	10.33
	t-ARK	**27.57**	-	-	-
	ARK	27.65	-	-	-
syn-tipr	uniform	61.61	29.14	32.47	-
	TransE	69.51	28.70	40.81	-
	ComplEx	63.96	23.15	40.81	-
	DistMult	67.51	26.70	40.81	-
	t-SAIL	26.30	-	11.13	15.17
	SAIL	27.14	-	9.90	17.24
	t-ARK	**23.34**	-	-	-
	ARK	23.48	-	-	-
syn-types	uniform	**36.02**	16.84	19.18	-
	TransE	48.26	19.05	29.21	-
	ComplEx	47.69	18.48	29.21	-
	DistMult	47.46	18.24	29.21	-
	t-SAIL	59.61	-	59.46	0.15
	SAIL	60.58	-	60.37	0.21
	t-ARK	59.79	-	-	-
	ARK	59.63	-	-	-
wd-movies	uniform	171.60	53.86	117.74	-
	TransE	208.60	51.39	157.21	-
	ComplEx	202.68	45.46	157.21	-
	DistMult	208.50	51.29	157.21	-
	t-SAIL	124.50	-	92.66	31.84
	SAIL	116.84	-	100.10	16.74
	t-ARK	114.49	-	-	-
	ARK	**98.19**	-	-	-
wd-articles	uniform	693.80	295.60	398.20	-
	TransE	910.65	280.67	629.98	-
	ComplEx	887.30	257.33	629.98	-
	DistMult	901.91	271.94	629.98	-
	t-SAIL	235.24	-	225.60	9.64
	SAIL	**199.55**	-	186.38	13.17
	t-ARK	224.25	-	-	-
	ARK	205.24	-	-	-

Table 5: We measure the compression quality for compressing graphs G. D_{KL} is only available for the VAE because it relies on the variational approximation, which is unique to this model. For the VAE, we compute an upper bound on the compression length (in bits). Probabilistic baseline (uniform, TransE, ComplEx, DistMult) results are from Thanapalasingam et al. (2023).

C Additional Figures

C.1 Architectural Details

Figure C.1 shows the architectural details of the t-SAIL model. Also Figure A.1 shows an example for conditioned generation.

Dataset	Model	Local Smoothness ↑	Global Consistency ↑	Flip Rate ↓	Avg Basin Length ↑
syn-paths	t-SAIL	0.75	0.36	0.20	4.54
	SAIL	0.74	0.14	0.33	2.87
syn-tipr	t-SAIL	0.99	0.98	0.09	8.61
	SAIL	0.93	0.69	0.10	8.03
syn-types	t-SAIL	0.82	0.60	0.12	6.80
	SAIL	0.92	0.73	0.20	4.47
wd-movies	t-SAIL	0.87	0.58	0.15	5.70
	SAIL	0.84	0.49	0.40	2.93
wd-articles	t-SAIL	0.81	0.55	0.14	5.37
	SAIL	0.82	0.57	0.17	3.46

Table 6: Latent space smoothness metrics for t-SAIL and SAIL with $\epsilon = 0.1$. Higher local/global smoothness indicates more continuous transitions. Lower flip rates suggest larger regions mapping to identical graphs.

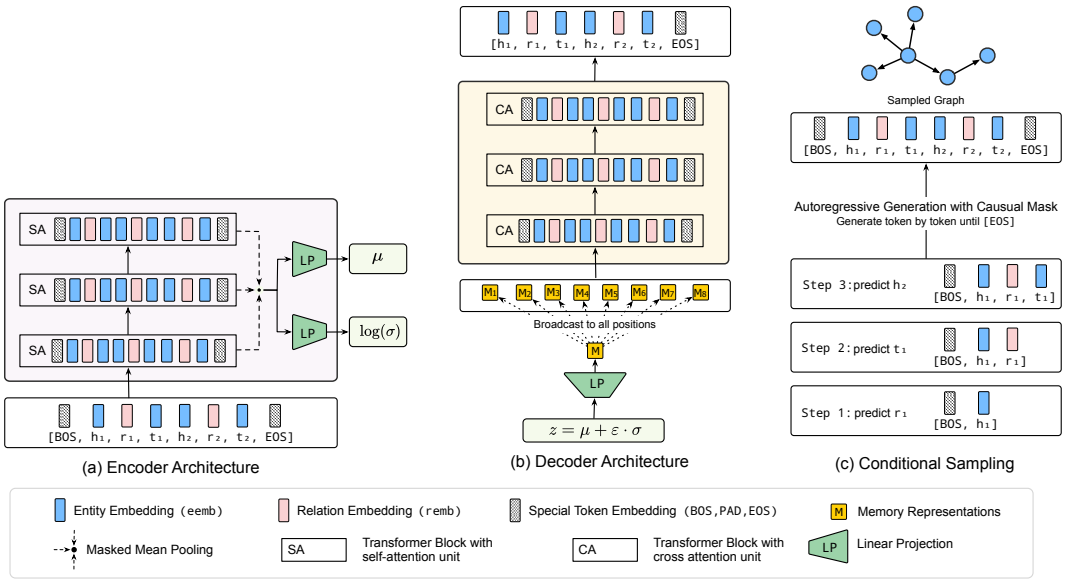


Figure C.1: t-SAIL has three main components: (a) an *Encoder* that processes linearized Knowledge Graph triple sequences $[\texttt{BOS}, h_1, r_1, t_1, h_2, r_2, t_2, \ldots, \texttt{EOS}]$ through self-attention (SA) blocks to produce latent distribution parameters $(\mu, \log \sigma)$, (b) a *Decoder* that uses cross-attention (CA) to condition on the sampled latent code z and autoregressively generates token sequences with causal masking, and (c) *Conditional Sampling* that demonstrates the step-by-step autoregressive generation process, predicting one token at a time until the `[EOS]` token is produced or the maximum sequence length is reached. The model uses a unified vocabulary embedding matrix spanning special tokens (`[BOS]`, `[PAD]`, `[EOS]`), entities (shown in blue), and relations (shown in pink), enabling sequential generation of Knowledge Graphs from learned latent representations.