ALTEGRAD 2025-2026 Data Challenge

Molecular Graph Captioning — graph- to-text generation

Selim-Antoine Lali

December 16, 2025

Abstract

This report summarizes our work on the ALTEGRAD Kaggle challenge *Molecular Graph Captioning*. The task is to map a molecular graph (atoms, bonds) to a natural-language caption describing the molecule. We first reproduce the teacher-provided baseline (embedding-based retrieval), then propose two successive retrieval improvements: (V2) an edge-aware GNN with contrastive learning and RoBERTa embeddings, and (V3) a CLIP-style model with a trainable text projection head, learnable temperature, larger batches, attention pooling, simple global graph features, and top-k medoid reranking. We report local validation metrics (BLEU, BERTScore) and Kaggle leaderboard scores showing consistent gains: 0.489 (baseline) $\rightarrow$ 0.513 (V2) $\rightarrow$ 0.604 (V3)...

1 Introduction

Graph-to-text generation aims to translate structured relational data into human-readable language. In cheminformatics, this corresponds to *molecular graph captioning*: given a molecule represented as a graph, predict a coherent caption describing key structural and functional aspects (e.g., functional groups, charge state, biological role). This multi-modal task sits at the intersection of Graph Representation Learning and Natural Language Processing (NLP).

In this report we focus on the **retrieval paradigm**: instead of generating novel text, the system retrieves the most relevant caption from a bank of training captions. This provides strong grammaticality and can be highly competitive when the dataset captions follow a templated style (as in this challenge).

2 Dataset and Problem Setting

Each molecule is provided as a `torch_geometric.data.Data` object containing:

- `x` $\in \mathbb{N}^{N\times 9}$: node (atom) features as 9 categorical indices (atomic number, chirality, degree, formal charge, number of H, radicals, hybridization, aromatic flag, ring flag).
- `edge_index` $\in \mathbb{N}^{2\times E}$: directed edges (two directions per undirected bond).
- `edge_attr` $\in \mathbb{N}^{E\times 3}$: bond categorical indices (bond type, stereo, conjugation flag).
- `id`: molecule identifier used for submission alignment.
- `description`: ground-truth caption (train/validation only).

The official Kaggle evaluation compares predicted captions against hidden ground-truth captions using a composite score derived from: (i) BLEU-4 F1 and (ii) BERTScore (RoBERTa-base).

3 Baseline pipeline (Retrieval by Embedding Similarity)

3.1 Overview

The baseline reframes captioning as a **shared-embedding retrieval** problem. Let $\mathcal{D} = \{(G_i, y_i)\}_{i=1}^{n}$ be the training set where G_i is a molecular graph and y_i its caption.

- A frozen **text encoder** produces a caption embedding $t_i \in \mathbb{R}^d$.
- A trainable **graph encoder** produces a graph embedding $g_i \in \mathbb{R}^d$.
- The graph encoder is trained to align g_i with t_i.
- At inference, for a test graph G, compute g and retrieve the closest caption embedding among the training set.

3.2 Text Embeddings (BERT CLS)

The baseline uses `bert-base-uncased` and takes the [CLS] embedding:

$$t_i = \mathrm{BERT}(y_i)_{\mathrm{CLS}} \in \mathbb{R}^{768}.$$

These embeddings are precomputed and stored (CSV).

3.3 Graph Encoder (GCN, topology-only)

In the teacher code, the GCN ignores both node features `x` and edge features `edge_attr`. All nodes are initialized with the same learnable vector $h_0 \in \mathbb{R}^h$:

$$H^{(0)} = \mathbf{1}h_0^\top \in \mathbb{R}^{N \times h}.$$

Then L GCN layers are applied, followed by a graph pooling:

$$H^{(\ell+1)} = \sigma\Big(\mathrm{GCNConv}\Big(H^{(\ell)}, E\Big)\Big), \qquad g = \mathrm{Proj}\Big(\mathrm{Pool}(H^{(L)})\Big), \qquad g \leftarrow \frac{g}{\|g\|_2}.$$

The teacher baseline uses `global_add_pool` and a final linear projection.

3.4 Training Objective (MSE alignment)

The baseline minimizes MSE between normalized embeddings:

$$\mathcal{L}_{\mathrm{MSE}} = \frac{1}{n}\sum_{i=1}^{n} \left\|\hat{g}_i - \hat{t}_i\right\|_2^2, \quad \hat{g}_i = \frac{g_i}{\|g_i\|_2},\ \hat{t}_i = \frac{t_i}{\|t_i\|_2}.$$

3.5 Inference (Nearest Neighbor retrieval)

Given a test molecule G, compute its embedding g and retrieve:

$$j^\star = \arg\max_{j \in \{1,\ldots,n\}} \cos(g, t_j) = \arg\max_j\ \hat{g}^\top \hat{t}_j,$$

and output the training caption $y_{j^\star}$.

3.6 Implementation sketch

Listing 1: Baseline retrieval core (simplified)

```
# g = mol_encoder(G)  # normalized
# T = stack(text_emb_train)  # normalized
sims = g @ T.T
j_star = sims.argmax()
y_pred = train_descriptions[j_star]
```

3.7 Experimental results (Baseline)

We reproduced the complete baseline pipeline (graph inspection, BERT embedding generation, GCN training, retrieval submission). The Kaggle score obtained on the test set was:

$$\textbf{Kaggle score (baseline)} = 0.489.$$

A typical observed issue is poor discrimination (many test molecules retrieving the same caption), consistent with a topology-only graph encoder.

4 Improved Retrieval Model V2

V2 targets the main limitations of the baseline: (i) ignoring chemical attributes, (ii) weak alignment loss, and (iii) text embeddings less aligned with BERTScore.

4.1 Key upgrades over baseline

- **Use atom features:** embed each categorical field in `x` with `nn.Embedding`.
- **Use bond features:** embed `edge_attr` and use an edge-aware message passing layer.
- **Edge-aware GNN:** switch from GCNConv to **GINEConv**.
- **Contrastive learning:** replace MSE with CLIP-style in-batch negatives (InfoNCE).
- **Text embeddings:** use **RoBERTa-base mean-pooled** embeddings to better correlate with BERTScore's backbone.
- **Evaluation:** validate like Kaggle by retrieving captions for validation graphs and computing BLEU + BERTScore.

4.2 Atom/Bond embedding

Let the atom categorical fields be $x_{i,1}, \ldots, x_{i,9}$ and bond fields be $e_{k,1}, e_{k,2}, e_{k,3}$. We embed and sum:

$$h_i^{(0)} = \sum_{m=1}^{9} \mathrm{Emb}^{(m)}(x_{i,m}), \qquad a_k = \sum_{m=1}^{3} \mathrm{Emb}_{\mathrm{edge}}^{(m)}(e_{k,m}).$$

4.3 Edge-aware message passing (GINE)

Using GINEConv (edge-conditioned MLP), for node i:

$$h_i^{(\ell+1)} = \mathrm{MLP}^{(\ell)}\Big((1+\epsilon)h_i^{(\ell)} + \sum_{j \in \mathcal{N}(i)} \sigma\big(h_j^{(\ell)} + a_{(j,i)}\big)\Big).$$

Then a graph-level embedding is obtained by pooling and projection, followed by ℓ_2 normalization.

4.4 Contrastive (CLIP-style) objective

With batch size B, graph embeddings g_i and text embeddings t_i:

$$\mathrm{logits}_{ij} = \frac{g_i^\top t_j}{\tau},$$

$$\mathcal{L} = \frac{1}{2}\left(\mathrm{CE}(\mathrm{logits}, \{1, \ldots, B\}) + \mathrm{CE}(\mathrm{logits}^\top, \{1, \ldots, B\})\right),$$

where τ is a temperature, and CE is cross-entropy with the diagonal as positives.

4.5 Experimental results (V2)

Local validation evaluation (retrieve-from-train on validation set) produced:

$$\text{BLEU} \approx 20.483, \qquad \text{BERTScore F1} \approx 0.8801.$$

Kaggle score improved from baseline:

$$\textbf{Kaggle score (V2)} = 0.513.$$

5 Improved Retrieval Model V3

V3 improves V2 with a stronger CLIP-style alignment and better retrieval decoding.

5.1 Key upgrades over V2

- **Train a text projection head:** learn a small MLP on top of frozen RoBERTa embeddings.
- **Learnable temperature:** optimize a logit scale parameter $\alpha = \exp(s)$.
- **More negatives:** increase batch size (e.g., 96 on CPU) to strengthen in-batch contrast.
- **Attention pooling:** use an attentional readout for graph embeddings.
- **Cheap global graph features:** concatenate simple statistics (atoms/bonds counts, aromatic/ring fractions, bond type fractions).
- **Medoid reranking:** retrieve top-k captions, then pick the most "central" candidate to increase caption typicality and BLEU.

5.2 Trainable text projection head

Let e_i be the frozen RoBERTa embedding for caption y_i. We learn:

$$t_i = \frac{f_\theta(e_i)}{\|f_\theta(e_i)\|_2},$$

where f_θ is a small MLP. This adapts the text space to be easier for the graph encoder to match, while keeping RoBERTa fixed.

5.3 Learnable temperature (logit scale)

Instead of a fixed τ, we learn a logit scale α:

$$\text{logits}_{ij} = \alpha\,(g_i^\top t_j), \qquad \alpha = \exp(s), \quad \alpha \in [1, 100],$$

optimized jointly with the graph encoder and the text projector.

5.4 Attention pooling and global features

Graph-level representation uses attentional aggregation:

$$g = \sum_{i=1}^{N} \beta_i h_i, \qquad \beta_i = \frac{\exp(u^\top h_i)}{\sum_j \exp(u^\top h_j)}.$$

We also compute a small vector of global features (8 scalars):

$\phi(G) = [\log(1+|V|),\ \log(1+|E|),\ \text{aromatic\_frac},\ \text{ring\_frac},\ \text{bond\_single\_frac},\ \text{bond\_double\_frac},\ \text{bond\_trip}$

embed it with an MLP, and concatenate to the pooled embedding before final projection.

5.5 Top-k medoid reranking

Given similarities between a query graph embedding g and the caption bank embeddings $\{t_j\}$:

1. Select indices $\mathcal{K}$ of the top-k captions by similarity.
2. Compute mean embedding $\bar{t} = \frac{1}{k}\sum_{j\in\mathcal{K}} t_j$, then normalize.
3. Choose the medoid:

$$j^\star = \arg\max_{j\in\mathcal{K}} \; t_j^\top \bar{t}.$$

This tends to select a more typical caption among close candidates, often improving BLEU while preserving semantic similarity.

5.6 Experimental results (V3)

Training stabilized strongly with learnable temperature and text projection (validation contrastive loss dropping from ≈ 1.58 to ≈ 0.24 at the best checkpoint).

Local Kaggle-like retrieval evaluation on validation:

$$\text{1-NN: BLEU} = 34.781, \quad \text{BERTScore F1} = 0.9109,$$

$$\text{Medoid@20: BLEU} = 37.553, \quad \text{BERTScore F1} = 0.9162.$$

On Kaggle test set, V3 achieved:

$$\textbf{Kaggle score (V3)} = 0.604.$$

Note on hardware. Training was performed on CPU due to instability encountered with PyTorch Geometric operations on Apple MPS for large graph batches in earlier attempts.

6 Summary Tables

6.1 Model comparison (retrieval paradigm)

Component	Baseline	V2	V3
Text encoder	BERT-base CLS, frozen	RoBERTa-base mean pool, frozen	RoBERTa-base mean pool, frozen + **text MLP projector**
Graph input	**Ignores** node/edge features	**Uses** 9 atom + 3 bond categorical features via embeddings	Same as V2 + **global graph features**
GNN layer	GCNConv (topology only)	**GINEConv** (edge-aware)	GINEConv + residual + LayerNorm
Pooling	global_add_pool	global_mean_pool (stable)	**attention pooling** (attentional aggregation)
Alignment loss	MSE(g, t)	**CLIP / InfoNCE** in-batch negatives	CLIP / InfoNCE + **learnable temperature**
Retrieval decoding	1-NN cosine	1-NN cosine	1-NN or **Top-k medoid** reranking
Batch negatives	limited	moderate	**larger batch** (e.g., 96)

Table 1: Summary of changes across models.

6.2 Performance comparison

Model	Local VAL BLEU	Local VAL BERTScore F1	Kaggle Test Score
Baseline	–	–	0.489
V2 (1-NN)	20.483	0.8801	0.513
V3 (1-NN)	34.781	0.9109	–
V3 (Medoid@20)	37.553	0.9162	0.604 (submission used retrieval decoding)

Table 2: Performance summary. Local validation scores are computed by retrieving captions for validation graphs from the training caption bank.

7 Short Terminal Excerpts

Listing 2: V3 local evaluation excerpt (validation)

```
Local Retrieval Eval on VAL (retrieve-from-TRAIN):
  1-NN:         BLEU=34.781 | BERTScoreF1=0.9109
  Medoid@5:     BLEU=37.269 | BERTScoreF1=0.9153
  Medoid@10:    BLEU=37.349 | BERTScoreF1=0.9158
  Medoid@20:    BLEU=37.553 | BERTScoreF1=0.9162
```

8 Attempted Improvements: V3 bis (Train+Val Bank, Larger Negatives)

After V3 (Kaggle score 0.604), we built a V3 bis variant to test several "low-risk" retrieval upgrades.

8.1 Changes with respect to V3

- **More in-batch negatives:** increased batch size to 128 and used gradient accumulation (`ACCUM_STEPS=2`), yielding an effective batch size of 256 for the CLIP-style contrastive loss.
- **Code-quality fix:** replaced deprecated `GlobalAttention` by `AttentionalAggregation` (no expected performance impact).
- **Keep attention pooling:** kept `POOLING="attn"`.
- **Medoid tuning:** evaluated additional medoid values $k \in \{25, 30, 35\}$ on validation.
- **Train+Val retrieval bank (inference only):** attempted to retrieve captions from the union of the training and validation caption bank to increase coverage.

8.2 Important implementation note: ID collisions in Train/Val

A subtle pitfall is that train and validation IDs are *not globally unique* (both start at 0). Therefore, a naive dictionary merge (e.g., `{**train, **val}`) causes collisions where validation entries overwrite training entries for the same keys. We corrected this by building the retrieval bank as *concatenated lists / stacked matrices* rather than merging dictionaries, ensuring the bank size truly becomes $31008 + 1000 = 32008$ captions.

8.3 Validation results (retrieve-from-train on validation)

On local validation, V3 bis slightly improved retrieval quality and confirmed that the optimal medoid is around $k = 10$ (larger k starts to degrade BLEU):

$$\text{1-NN: BLEU} = 36.415, \quad \text{BERTScore F1} = 0.9137,$$

$$\text{Medoid@10: BLEU} = 38.757, \quad \text{BERTScore F1} = 0.9173.$$

(For comparison, V3 best medoid on validation was around $k = 20$ with BLEU 37.553 and BERTScore F1 0.9162.)

8.4 Kaggle leaderboard result and conclusion

Despite the local validation gains, V3 bis did not improve leaderboard performance:

$$\textbf{Kaggle score (V3)} = 0.604, \qquad \textbf{Kaggle score (V3 bis, train+val bank)} \approx 0.602\text{–}0.603.$$

This suggests we are close to a **retrieval ceiling** for this setup (same caption bank, same fixed text encoder backbone, same retrieval decoding family), and further improvements may require either ensembling or moving beyond pure retrieval (e.g., retrieve-then-edit or fully generative decoding).

8.5 Updated score table

Model	Local VAL BLEU	Local VAL BERTScore F1	Kaggle Test Score
Baseline (teacher)	–	–	0.489
V2 (1-NN)	20.483	0.8801	0.513
V3 (1-NN)	34.781	0.9109	–
V3 (Medoid@20)	37.553	0.9162	0.604
V3 bis (1-NN)	36.415	0.9137	–
V3 bis (Medoid@10)	38.757	0.9173	0.602

Table 3: Performance summary (retrieval paradigm). Local validation scores are computed by retrieving captions for validation graphs from the training caption bank.

9 Results & Plots

This section summarizes the empirical behavior of our retrieval models (V2, V3, V3 bis) through (1) leaderboard evolution, (2) local validation metrics, (3) training dynamics, and (4) the effect of the medoid top-k reranking used to select more "typical" captions.

9.1 Leaderboard progression

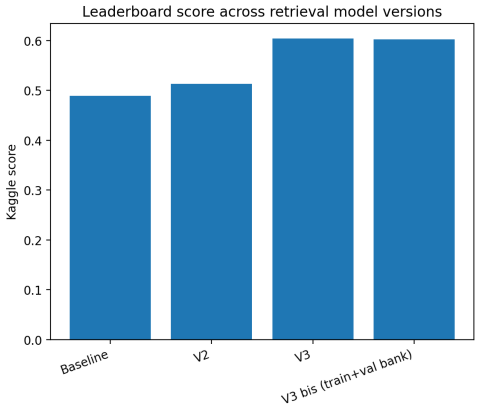
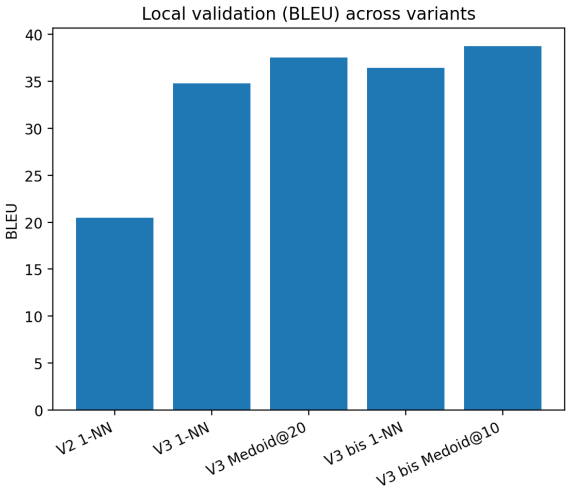


Figure 1: Kaggle leaderboard score across retrieval model versions.

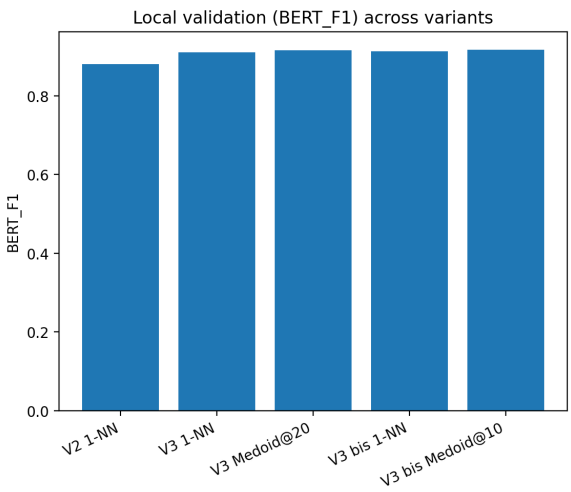
Figure 1 shows the Kaggle score progression across retrieval versions. The largest jump occurs from V2 to V3, consistent with the introduction of a trainable text projection head, learnable temperature (logit scale), attention pooling, and improved decoding (medoid reranking).

9.2 Local validation retrieval quality

We evaluate locally in a Kaggle-like manner by treating the validation set as "test": for each validation molecule, we retrieve a caption from the training bank and compute BLEU and BERTScore against the validation ground truth.



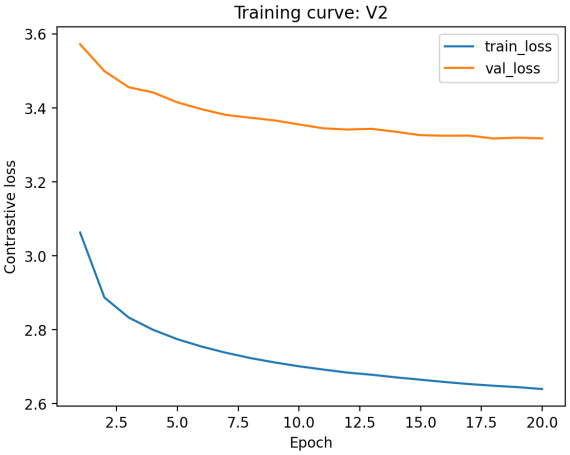
(a) Local validation BLEU across key retrieval variants.



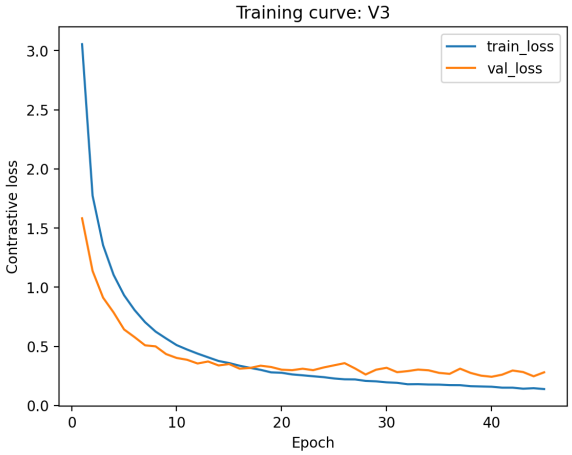
(b) Local validation BERTScore F1 across key retrieval variants.

9.3 Training dynamics: contrastive loss curves

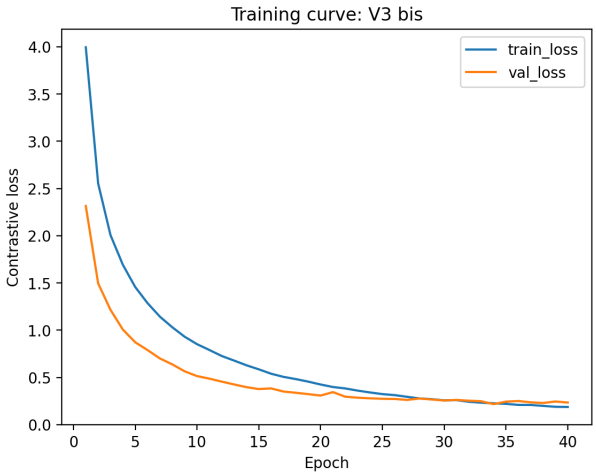
Figures 3a, 3b, and 3c show the training and validation contrastive losses (CLIP-style) over epochs. V3 and V3 bis converge substantially faster and lower than V2, reflecting better alignment and stronger in-batch negatives.



(a) Training/validation contrastive loss for V2.



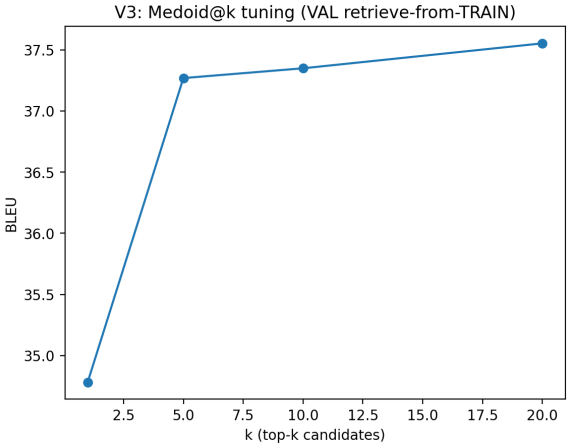
(b) Training/validation contrastive loss for V3.



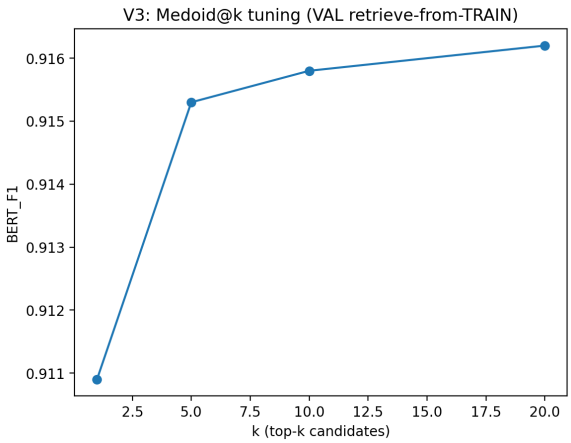
(c) Training/validation contrastive loss for V3 bis (larger effective batch via accumulation).

9.4 Choosing the medoid top-k: validation tuning curves

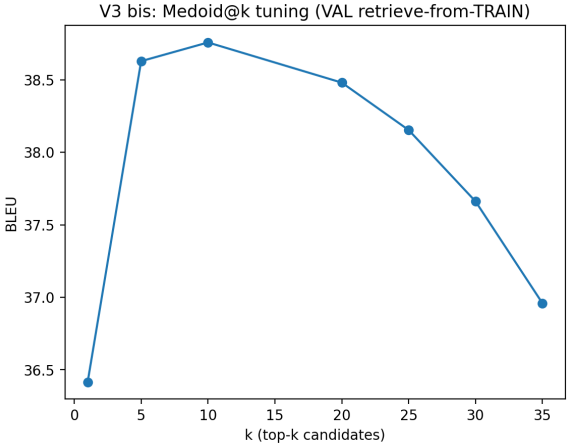
We tune the decoding strategy by evaluating the medoid reranking parameter k on validation. Figures 4a–4d illustrate that performance peaks for small-to-moderate k values; overly large k tends to reduce BLEU (captions become less specific). This provides an empirical justification for selecting $k = 20$ in V3 and $k = 10$ in V3 bis.



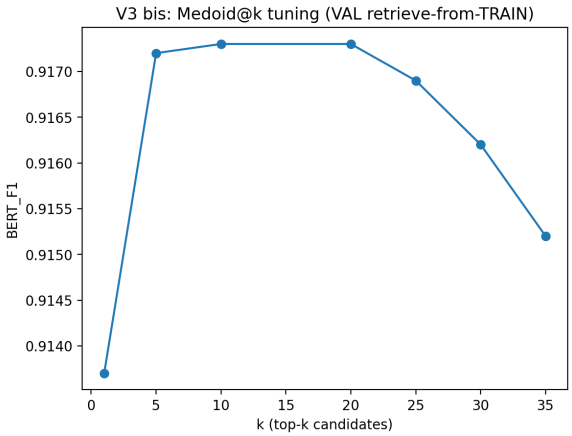
(a) V3: Medoid@k tuning on validation (BLEU).



(b) V3: Medoid@k tuning on validation (BERTScore F1).



(c) V3 bis: Medoid@k tuning on validation (BLEU).



(d) V3 bis: Medoid@k tuning on validation (BERTScore F1).

10 Conclusion and Next Steps

We demonstrated that strong improvements are possible within a retrieval-only formulation by: (1) leveraging atom/bond features with an edge-aware GNN, (2) switching to contrastive alignment, (3) adapting text embeddings with a projection head and learnable temperature, and (4) improving decoding with medoid reranking. Our best retrieval system (V3) substantially improves the Kaggle score to 0.604.

Next steps (beyond this report) include exploring true graph-to-text generation (e.g., GNN encoder + Transformer decoder), and/or hybrid approaches (retrieve-then-edit), while keeping evaluation aligned with BLEU and BERTScore.

11 Graph-to-Text Generation Models (V0–V2)

After exploring the retrieval paradigm and observing a performance ceiling, we investigated *true graph-to-text generation*, i.e., learning a conditional language model $p_\theta(y \mid G)$ that directly produces a caption y from a molecular graph G. This section describes the generation architectures we implemented, the motivations behind each upgrade, and the experimental results obtained on validation and on the Kaggle leaderboard.

11.1 Problem Formulation

Each molecule is represented by a graph $G = (V, E)$ with node attributes $x_v \in \mathbb{Z}^9$ and edge attributes $e_{uv} \in \mathbb{Z}^3$ (categorical indices). The goal is to generate a natural language description $y = (y_1, \ldots, y_T)$.

In all generative models, we optimize a token-level negative log-likelihood (teacher forcing):

$$\mathcal{L}(\theta) \;=\; -\sum_{t=1}^{T} \log p_\theta(y_t \mid y_{<t},\, G)\,, \tag{1}$$

with early stopping based on validation loss. At inference, a decoding algorithm (beam search) produces $\hat{y}$, which is scored on Kaggle using a composite of BLEU-4 and BERTScore (RoBERTa-based).

11.2 Common Building Blocks

Graph encoder. Across models, the molecular graph is encoded using an edge-aware message passing network based on `GINEConv`. Categorical node and edge features are embedded with learned embedding tables and aggregated into continuous representations.

Conditioning a Transformer decoder. The key design choice is how graph information is injected into a pretrained language model (LM). We explored *prefix conditioning*: a graph encoder produces a sequence of continuous "latent" tokens $\mathbf{Z} \in \mathbb{R}^{P \times d}$, which are prepended to the decoder inputs, so that the LM attends to $\mathbf{Z}$ during generation. Concretely, let $\mathbf{E}(x)$ be the token embedding layer of the LM for text inputs. We form:

$$\mathbf{H}_{\text{enc}} \;=\; [\,\mathbf{Z}\,;\; \mathbf{E}(x_{1:L})\,], \tag{2}$$

with a corresponding attention mask that enables the model to attend to both the graph prefix and text tokens.

Decoding. We used beam search with constraints to reduce repetition: `num_beams`, `max_new_tokens`, `length_penalty`, and `no_repeat_ngram_size`. We also performed a small grid search on these parameters (Section 11.5.1).

11.3 Model V0: GNN Prefix + GPT-2 (Baseline Generative)

Idea. Our first generative baseline (V0) uses a graph encoder to produce a small prefix that conditions a GPT-2 decoder. The decoder is trained end-to-end with teacher forcing on (graph, caption) pairs.

Decoder. We used `GPT2LMHeadModel` (`gpt2`) as the causal LM. The model is conditioned on a learned prefix of length P that summarizes the graph.

Results. Training converged smoothly (validation loss ≈ 0.63). Local generation evaluation on validation achieved:

- BLEU (sacrebleu): **24.500**
- BERTScore F1 (RoBERTa): **0.8915**

Kaggle test score: **0.545**.

11.4 Model V1: Stronger GPT-2 Conditioning + Decoding Tweaks

Motivation. V0 showed the approach works but was still far below the retrieval ceiling. We increased training stability and explored a stronger conditioning setup and decoding parameters.

Decoder. We kept GPT-2 as decoder and used beam search. Despite improved training loss, validation generation metrics degraded (likely due to decoding mismatch and output truncation).

Results. Validation decoding configuration (example): `num_beams=5`, `max_new_tokens=140`, `length_penalty=1.0`, `no_repeat_ngram_size=3`.

- BLEU: **23.000**
- BERTScore F1: **0.8831**

Kaggle test score: **0.520**.

11.5 Model V1.1: Improved Prefix Architecture + Better Alignment

Motivation. The main limitation in V0/V1 is the information bottleneck: a small prefix must carry all molecular information. We therefore improved the prefix mechanism and training setup, leading to a large jump in validation metrics.

Model. V1.1 keeps a transformer decoder but improves: (i) the graph encoder/prefix pathway to reduce information loss, and (ii) training stability. This model produced substantially better validation BLEU and BERTScore.

Results. Training reached a best validation loss around **0.6057**. Local validation evaluation:

- BLEU: **32.158**
- BERTScore F1: **0.9140**

Kaggle test score (default decode): **0.602**.

11.5.1 Decoding Tuning for V1.1

Motivation. We observed that `max_new_tokens` has a strong impact: too small values truncate the templated chemical descriptions, harming BLEU and sometimes BERTScore. We ran a grid search on beam width, length penalty, no-repeat constraint, and maximum generation length.

Best configuration found.

- `num_beams=5`
- `max_new_tokens=130`
- `length_penalty=1.1`
- `no_repeat_ngram_size=4`

This achieved on validation: BLEU= 37.288, BERTScore F1= 0.9188 (proxy score = 0.6458), and yielded a Kaggle test score of **0.619**. This indicates that decoding hyperparameters are a critical part of the pipeline.

11.6 Model V1.2: BART-large with Larger Latent Tokens (Attempt)

Motivation. Because BART is a strong conditional generation model, we attempted to increase capacity by switching to `facebook/bart-large` and increasing the number of latent/prefix tokens.

Outcome. Despite using an A100 GPU and gradient checkpointing, training became unstable and generalization degraded. The best Kaggle score obtained was **0.496**, significantly worse than V1.1. This suggests that the "soft-prefix-only" conditioning is not sufficient to control a much larger decoder without additional architectural constraints, longer training, or better optimization schedules.

11.7 Model V2: Retrieve-Augmented Generation (RAG) + Graph Prefix

Motivation. V1.1 demonstrated that generation can reach ~ 0.62 with careful decoding, but fully generative approaches remain hard. We thus attempted a retrieve-augmented generation approach: use a strong graph→text retriever to fetch top-k candidate captions and let the generator refine/compose the final caption, while still conditioning on the graph.

Architecture. V2 has two stages:

1. **Retriever (CLIP-style):** learn a graph encoder $f_\theta(G)$ aligned to fixed RoBERTa text embeddings $g(\text{text})$ using a contrastive loss with in-batch negatives.
2. **RAG Generator:** feed the retrieved captions as encoder text context and prepend a learned graph prefix to the encoder embeddings. A BART-large decoder then generates the final caption.

Training. The retriever trained well (train loss decreased from 4.60 to 0.73 over 15 epochs). The RAG generator achieved a best validation loss of **0.7104** around epoch 9.

Results. Despite promising training curves, the Kaggle test score was **0.582**, worse than V1.1. A plausible explanation is that the generator overfits to the retrieved templates or that the retrieval noise introduces lexical artifacts which hurt BLEU on the hidden test split. This indicates that RAG is not automatically beneficial unless retrieval quality and generator fusion are carefully balanced (e.g., better prompt formatting, stronger copy-control, or reranking).

11.8 Comparison of Generation Models

Model	Key idea / differences
V0	Graph encoder → soft prefix → GPT-2 decoder; teacher forcing; baseline generative pipeline.
V1	GPT-2 generation with additional stabilization and decoding; did not improve generalization.
V1.1	Improved conditioning/prefix mechanism and training stability; large gains in BLEU/BERTScore; strong decoding tuning.
V1.2	Switch to BART-large + more prefix tokens; training unstable and poor leaderboard score.
V2	Retrieve-Augmented Generation: (graph→text retrieval) + BART-large conditioned on retrieved captions + graph prefix.

Table 4: Summary of architectural changes across graph-to-text generation models.

Model	Val BLEU	Val BERTScore F1	Kaggle Score
V0	24.500	0.8915	0.545
V1	23.000	0.8831	0.520
V1.1 (default decode)	32.158	0.9140	0.602
V1.1 (tuned decoding)	37.288	0.9188	**0.619**
V1.2	–	–	0.496
V2 (RAG + prefix)	–	–	0.582

Table 5: Generation results. "–" indicates metrics not computed or not retained for that run.

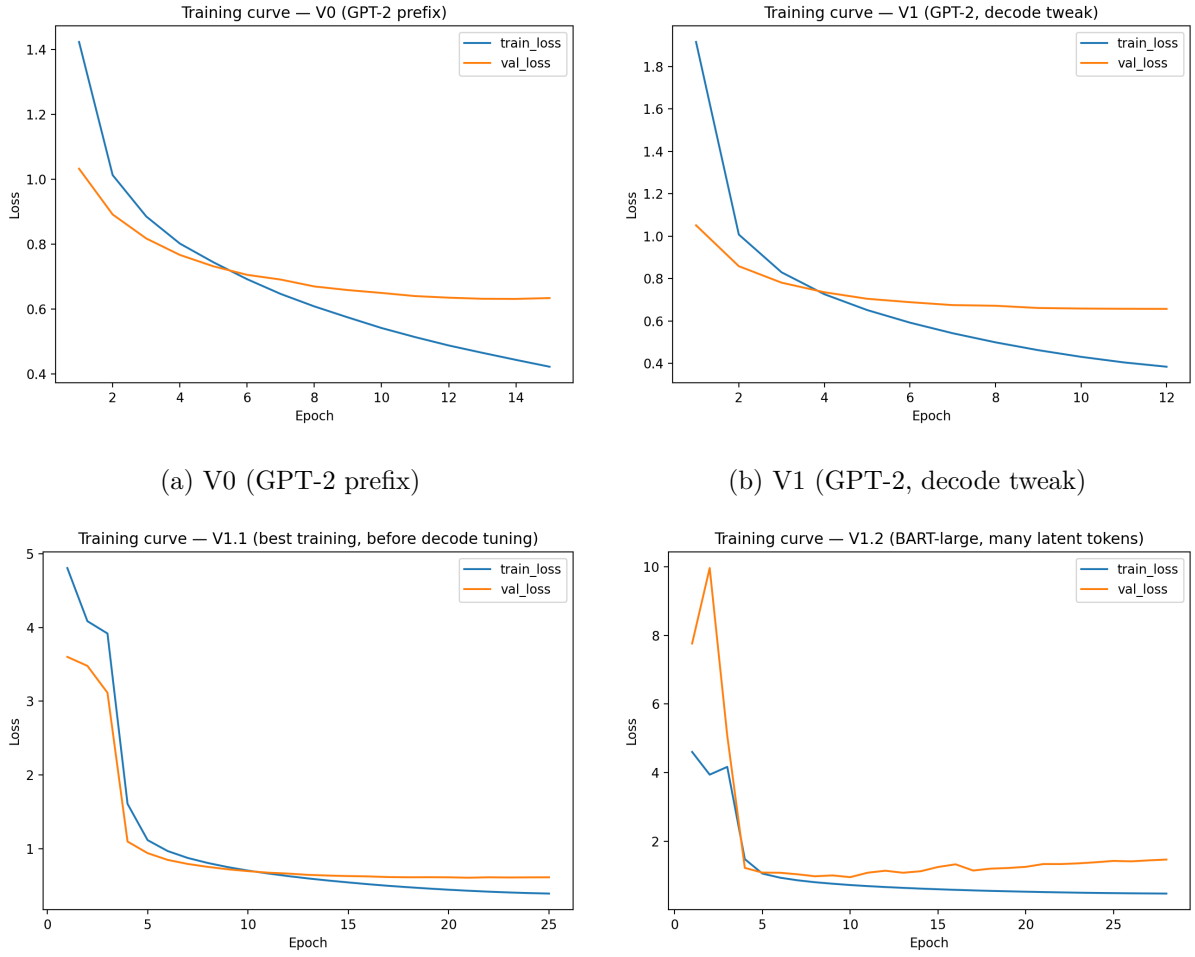
Key takeaway. For generation, the most impactful improvement we observed was **decoding optimization** (V1.1), especially the maximum generation length, which strongly affects BLEU by preventing truncation of templated chemical descriptions. Naively increasing model size (BART-large) or adding retrieval context (RAG) did not automatically improve leaderboard performance in our current setup.

12 Results & Plots (Graph-to-Text Generation)

This section summarizes the empirical behavior of our generation models (V0, V1, V1.1, V1.2, V2) on the validation split (local BLEU/BERTScore) and on Kaggle (hidden test split). We also visualize training dynamics (train/val loss curves), compare final scores across versions, and analyze the impact of decoding hyperparameters (beam search, maximum generated length, repetition constraints) for the best-performing GPT-2-based model.

12.1 Training dynamics

Figures 5–7 show the training/validation loss trajectories across model variants. While loss is not directly comparable across different decoder families, it remains useful for (i) early stopping, and (ii) diagnosing instability or overfitting within a given architecture.



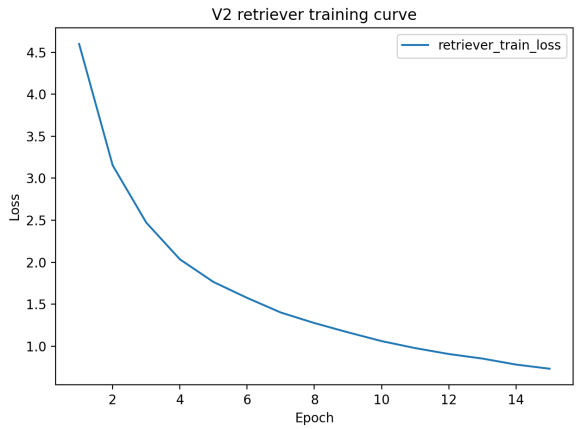
(a) V0 (GPT-2 prefix)

(b) V1 (GPT-2, decode tweak)

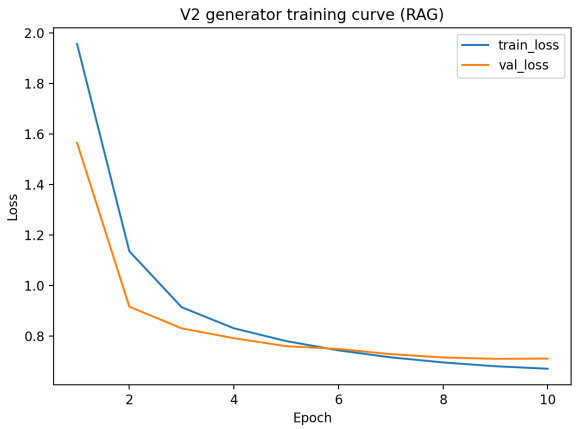
(c) V1.1 (best training)

(d) V1.2 (BART-large)

Figure 5: Training curves (train/val loss) for generation models.



(a) V2 retriever training



(b) V2 generator training (RAG)

Figure 6: V2 (RAG) has a two-stage pipeline: a retriever trained with contrastive loss, then a conditional generator trained with teacher forcing.

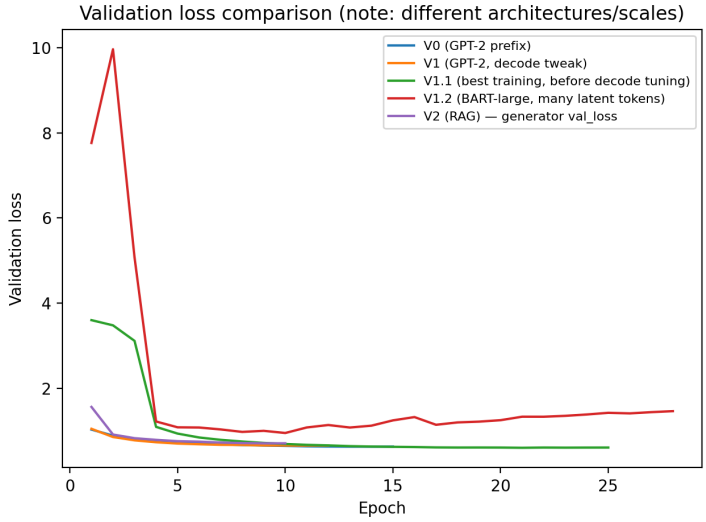
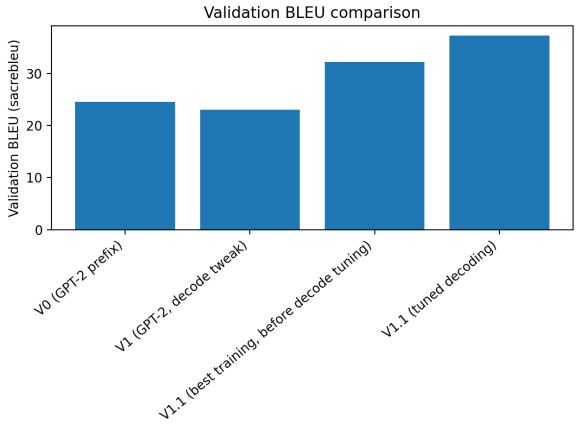


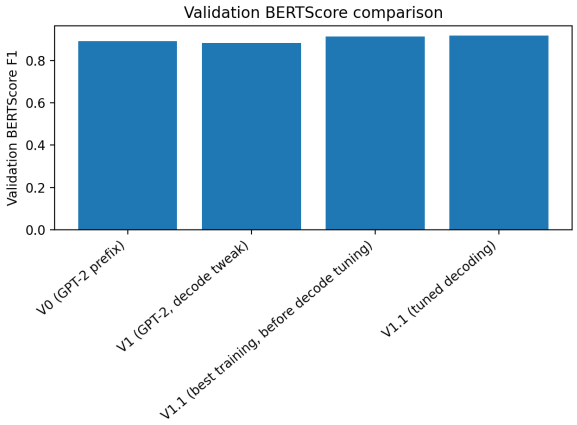
Figure 7: Overlay of validation losses across runs (for reference; note that absolute values may not be directly comparable across architectures).

12.2 Validation metrics and Kaggle scores

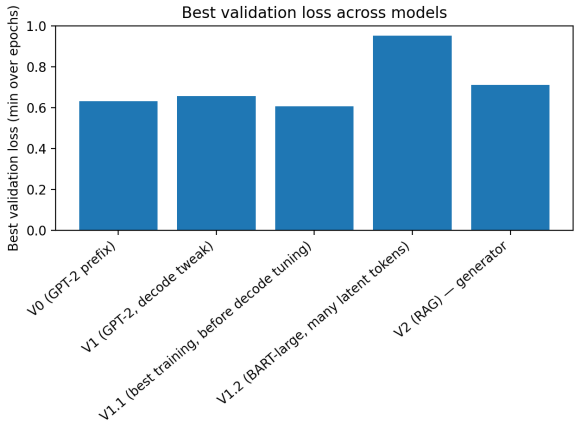
Figure 8 summarizes local validation metrics (BLEU and BERTScore F1) whenever available, and Figure 9 reports Kaggle scores on the hidden test set. The strongest jump among GPT-2-based models is achieved by improving decoding, especially by increasing `max_new_tokens` (reducing truncation) and controlling repetition (no-repeat n-gram constraint).



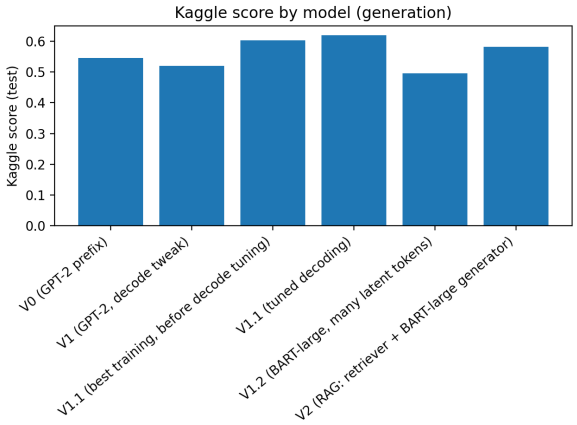
(a) Validation BLEU



(b) Validation BERTScore F1



(c) Best validation loss



(d) Kaggle score (test)

Figure 8: Summary metrics for Graph-to-Text generation runs.

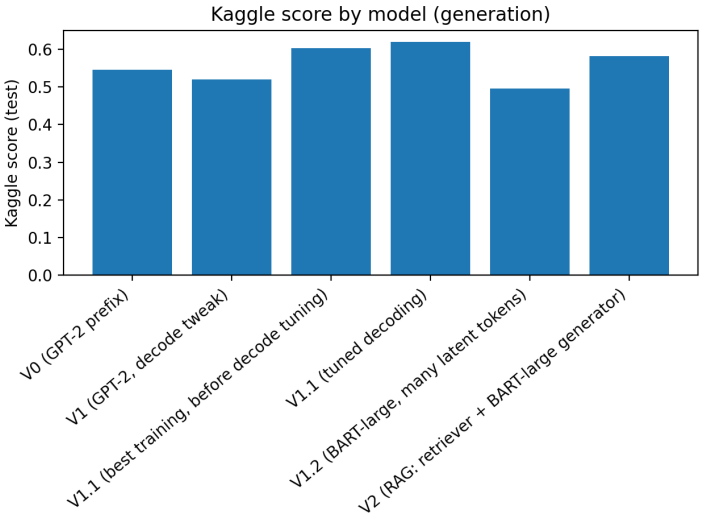
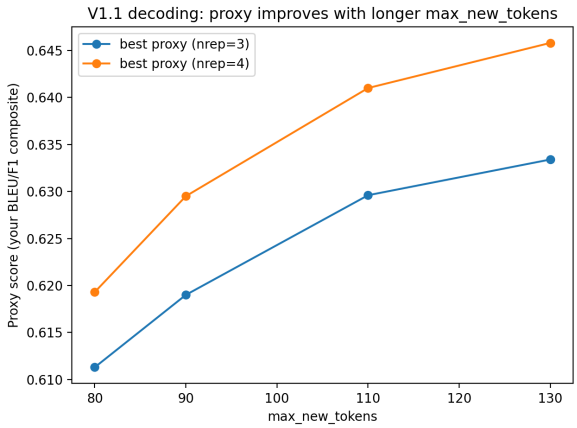


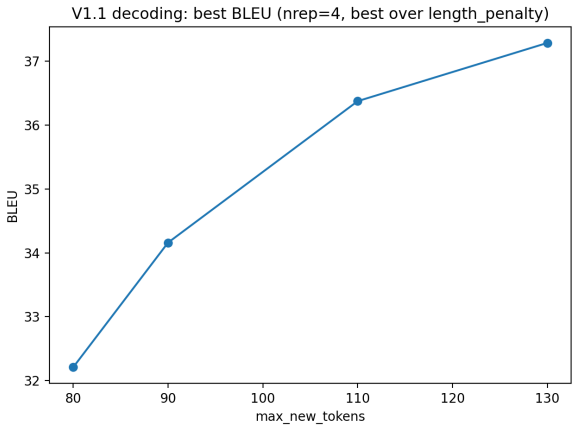
Figure 9: Kaggle scores (hidden test split) for generation models. V1.1 with tuned decoding achieves the best score among our pure generation runs (0.619).

12.3 Decoding hyperparameter study (V1.1)

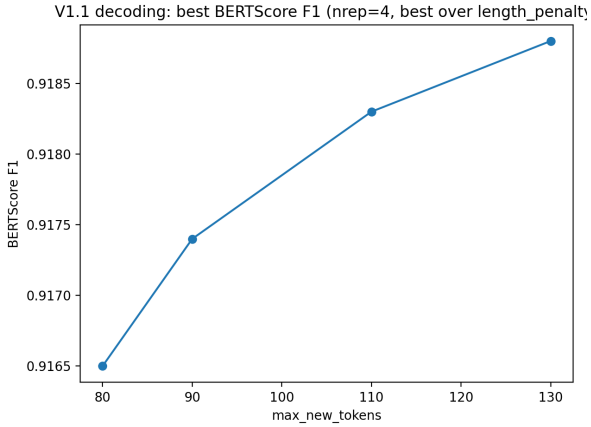
We performed a grid search over beam size, maximum generation length, length penalty, and no-repeat n-gram constraints. Figures 10–?? show that increasing `max_new_tokens` from 80 to 130 consistently improves both BLEU and BERTScore (and thus the proxy score), strongly suggesting that shorter decoding settings were truncating valid captions.



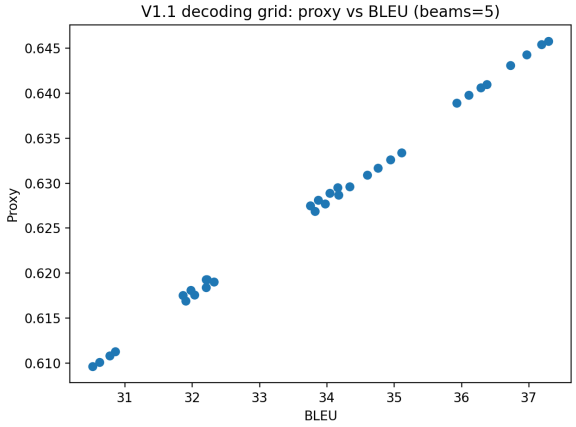
(a) Proxy vs `max_new_tokens`



(b) Best BLEU vs `max_new_tokens`



(c) Best BERTScore F1 vs `max_new_tokens`



(d) Proxy vs BLEU (grid points)

Figure 10: Decoding study for V1.1: impact of generation length and repetition constraints.

Best decoding configuration (V1.1). The best-performing configuration on validation in our sweep was:

`num_beams` $= 5$, `max_new_tokens` $= 130$, `length_penalty` $= 1.1$, `no_repeat_ngram_size` $= 4$,

which led to a Kaggle score of 0.619 on the hidden test split.

12.4 Takeaways

Overall, the strongest improvements in the generation setting came from: (i) stabilizing training and conditioning the decoder effectively on graph information (V1.1), and (ii) carefully tuning decoding, especially generation length, to avoid truncation and preserve templated factual details that are rewarded by BLEU and BERTScore.

References

[1] Carl Edwards, Tuan Lai, Kevin Ros, Garrett Honke, Kyunghyun Cho, and Heng Ji. Translation between Molecules and Natural Language. In *EMNLP*, 2022.

[2] Sixing Ji et al. A survey on graph-to-text generation: Tasks, methods, and challenges. *arXiv:2203.04803*, 2022.

[3] Shuzhi Yuan and Michael Faerber. Evaluating generative models for graph-to-text generation. In *RANLP*, 2023.