Z-GED Model Architecture Summary

Graph-VAE Circuit Topology Generation for RLC Filters

Project Technical Brief

February 7, 2026

Executive Summary

Z-GED is a variational graph generative model that maps RLC circuits to an 8-dimensional latent space and decodes latent codes into circuit topologies. The system combines:

- a component-aware GNN encoder for graph structure and element semantics,
- a hierarchical latent decomposition for topology, component-structure, and transfer-function factors,
- an autoregressive transformer decoder for node and edge-component generation.

In repository-reported validation results, the model reaches near-perfect structural reconstruction on the benchmark split (72 validation circuits), suggesting strong in-distribution fidelity.

1 Problem Formulation

Given a circuit graph $G = (V, E)$ with typed nodes and impedance-derived edge attributes, the model learns:

$$q_\phi(z \mid G, P, Z), \quad p_\theta(G \mid z),$$

where P and Z are variable-length pole/zero sets, and $z \in \mathbb{R}^8$.

Generation can be done by:

- random latent sampling $z \sim \mathcal{N}(0, I)$,
- encoding an existing graph and decoding z,
- interpolation between latent codes.

2 Input Representation

2.1 Node Features

Each node is one-hot encoded with 4 categories:

$$[\text{is\_GND}, \text{is\_VIN}, \text{is\_VOUT}, \text{is\_INTERNAL}].$$

2.2 Edge Features

Each directed edge uses 3 features:

$$[\log_{10} R,\ \log_{10} C,\ \log_{10} L],$$

with 0 used as the sentinel for "component absent" on that edge. Component-presence masks are derived inside `ImpedanceConv` via thresholding.

2.3 Auxiliary Physical Descriptors

Poles and zeros are provided as variable-length sets of 2D real-imaginary vectors and encoded via DeepSets.

3 Encoder Architecture (HierarchicalEncoder)

3.1 Stage 1: Component-Aware Message Passing

The encoder uses a 3-layer ImpedanceGNN built from `ImpedanceConv`. Messages are component-specific:

$$m_{ij} = \mathbf{1}_{R,ij}\, f_R([h_j, \log_{10} R_{ij}]) + \mathbf{1}_{C,ij}\, f_C([h_j, \log_{10} C_{ij}]) + \mathbf{1}_{L,ij}\, f_L([h_j, \log_{10} L_{ij}]),$$

where $\mathbf{1}_{R,ij}, \mathbf{1}_{C,ij}, \mathbf{1}_{L,ij}$ are internally derived presence indicators. The resulting message is then weighted by learned attention and aggregated.

3.2 Stage 2: Three-Branch Latent Heads

The encoder forms three latent branches:

- **Topology branch** ($z_{\text{topo}} \in \mathbb{R}^2$): global mean+max pooled node embeddings.
- **Structure/values branch** ($z_{\text{struct}} \in \mathbb{R}^2$): concatenated embeddings of GND/VIN/VOUT.
- **Pole-zero branch** ($z_{\text{pz}} \in \mathbb{R}^4$): DeepSets encodings of poles and zeros.

Concatenation gives $z = [z_{\text{topo}}, z_{\text{struct}}, z_{\text{pz}}] \in \mathbb{R}^8$, with VAE sampling:

$$z = \mu + \exp\left(\frac{1}{2}\log\sigma^2\right) \odot \epsilon, \quad \epsilon \sim \mathcal{N}(0, I).$$

4 Decoder Architecture (SimplifiedCircuitDecoder)

4.1 Latent-to-Context Projection

The latent code is projected through an MLP with LayerNorm to a hidden state ($d = 256$ in production config).

4.2 Node Count Prediction

Node count is classified directly from z over classes $3, 4, \ldots,$ max_nodes.

4.3 Autoregressive Node Decoder

Nodes are generated sequentially using a transformer decoder:

$$p(v_i \mid z, v_{<i}, N).$$

The query includes latent context, position embedding, and total-length embedding.

4.4 Autoregressive Edge-Component Decoder

For ordered edge pairs (i, j) with $j < i$, the model predicts an 8-class variable:

$$c_t \in \{0, 1, \ldots, 7\},$$

where 0 means no edge, and 1–7 encode {R, C, L, RC, RL, CL, RCL}. The factorization is:

$$p(C \mid z, V) = \prod_{t=1}^{T} p(c_t \mid z, V, c_{<t}),$$

implemented with causal self-attention and previous edge-token embeddings (teacher forcing in training, autoregressive feedback at inference).

5 Training Objective

The total loss is:

$$\mathcal{L} = \lambda_n \mathcal{L}_{\text{node-type}} + \lambda_c \mathcal{L}_{\text{node-count}} + \lambda_e \mathcal{L}_{\text{edge-comp}} + \lambda_{conn} \mathcal{L}_{\text{connectivity}} + \lambda_{kl} \mathcal{L}_{KL}.$$

Repository training configuration:

$$(\lambda_n, \lambda_c, \lambda_e, \lambda_{conn}, \lambda_{kl}) = (1.0,\ 5.0,\ 2.0,\ 5.0,\ 0.01),$$

with KL warmup during early epochs.

5.1 Connectivity Regularization

Connectivity loss penalizes:

- disconnected VIN,
- disconnected VOUT,
- globally weak graph connectivity,
- isolated non-mask nodes.

6 Model Size and Data Regime

Reported architecture scale:

- Encoder: 83,411 parameters.
- Decoder: 7,698,901 parameters.

Reported dataset and split:

- 360 circuits total, 6 filter families (60 per family).
- 288 train / 72 validation.

7 Empirical Results (Repository-Reported)

Metric	Training	Validation
Total loss	0.95	1.03
Node-count accuracy	100%	100%
Edge-existence accuracy	100%	100%
Component-type accuracy	100%	100%

Additional reported latent-space behavior:

- clear clustering by filter family in $z_{0:4}$,
- smooth topology transitions under latent interpolation,
- non-trivial but weaker activity in transfer-function dimensions $z_{4:8}$.

8 Strengths and Caveats for Academic Discussion

8.1 Strengths

- Physically informed edge featurization and component-aware message passing.
- Explicit autoregressive dependence for edge decisions via causal attention.
- Interpretable latent decomposition aligned with topology and spectral descriptors.
- Good in-distribution reconstruction and generation fidelity in project benchmarks.

8.2 Caveats

- Benchmark appears structurally templated; perfect scores may reflect limited topology entropy.
- Transfer-function branch ($z_{4:8}$) is only weakly supervised by current objectives.
- Random latent sampling still yields invalid circuits in reported novelty sweeps.
- Specification-conditioned generation in scripts is partly driven by latent-neighbor retrieval, not solely by an end-to-end conditional decoder.

9 Recommended Next Experiments

- Add auxiliary supervision from (f_c, Q) or pole/zero reconstruction losses to activate $z_{4:8}$.
- Evaluate out-of-distribution robustness with wider component ranges and unseen topologies.
- Compare against non-autoregressive graph decoders and conditional baselines.
- Validate generated circuits via SPICE-based functional metrics, not only structural matching.

Code Pointers

Core files in this repository:

- `ml/models/encoder.py`
- `ml/models/gnn_layers.py`
- `ml/models/decoder.py`
- `ml/models/decoder_components.py`
- `ml/models/node_decoder.py`
- `ml/losses/circuit_loss.py`
- `ml/losses/connectivity_loss.py`
- `ml/data/dataset.py`