Graph Tokenization for Bridging Graphs and Transformers

Bringing graph-structured data into the Transformer and LLM ecosystem

Zeyuan Guo, Enmao Diao, Cheng Yang, Chuan Shi · BUPT / DreamSoul · ICLR 2026

OUTLINE

1. **Motivation** : why graph data does not fit the Transformer toolbox
2. **Preliminaries** : labeled graphs, what a serialization must guarantee, and BPE
3. **Method** : turning a graph into tokens that a standard Transformer can read

1.1 Every modality joined the Transformer era through a tokenizer

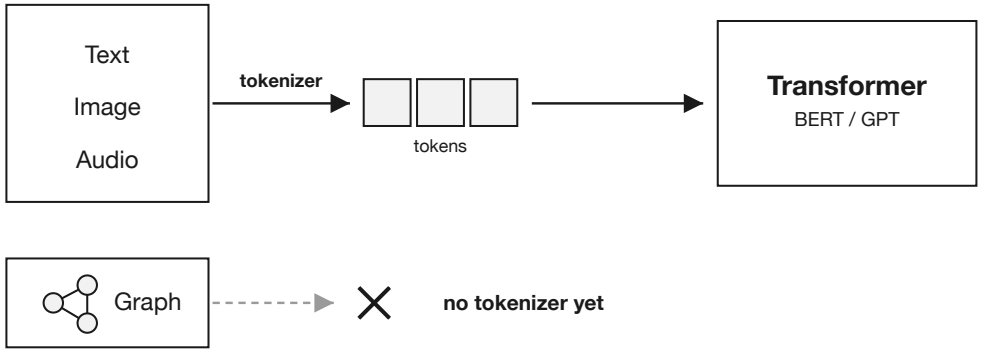

Figure 1: the tokenizer is the bridge. Text, images, and audio each cross into the Transformer by being tokenized; graphs have no standard tokenizer, so they cannot make the jump.

- Over the last few years, one architecture became the default across machine learning: the **Transformer**, and the LLMs built on it.
- Each modality made the jump the same way: a **tokenizer** turns raw input into discrete **tokens**, subword units for text, **patches** for a Vision Transformer, codec tokens for audio.
- Once the data is tokens, it inherits the whole ecosystem: pretraining, scaling, and one shared toolbox.

1.2 One kind of data sat the revolution out: graphs

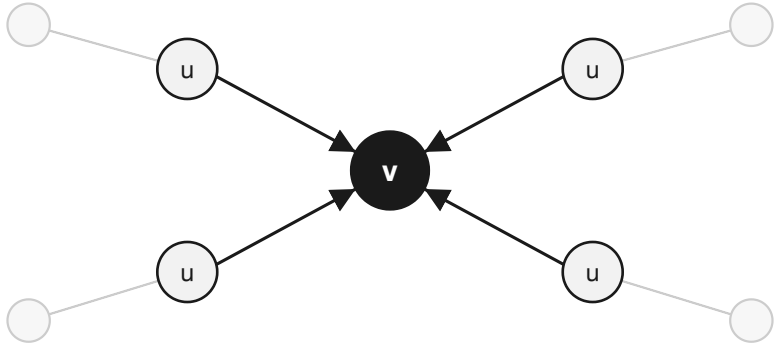

Figure 2: message passing. A Graph Neural Network updates each node from its neighbours: the irregular-domain analogue of a convolution's receptive field.

- One common data type never got that tokenizer: the **graph**.
- Graphs are not a niche, even in vision: molecules, social and citation networks, scene graphs, 3D meshes, point clouds and skeletons.
- We do model them, but with a **separate toolbox**, the **Graph Neural Network**: message passing along edges, the graph cousin of a convolution.
- So graphs sit on their own island, outside the Transformer and LLM ecosystem.

1.3 Two ways to put Transformers on graphs, both compromise

(a) Change the architecture

- Bake attention into GNNs to build specialized **Graph Transformers**.
- Needs graph-specific designs that **diverge from standard sequence models** and their ecosystem.
- Every advance in sequence modeling has to be re-implemented for graphs.

(b) Embed into continuous vectors

- Project graph structure into the embedding space of a pretrained model.
- Lossy or unstable representations; performance hinges on **cross-modal alignment**.
- Information is discarded before the model even starts.

> **Neither gives a clean, lossless, off-the-shelf interface. That gap is what a real graph tokenizer should fill.**

1.4 Why graphs resist tokenization

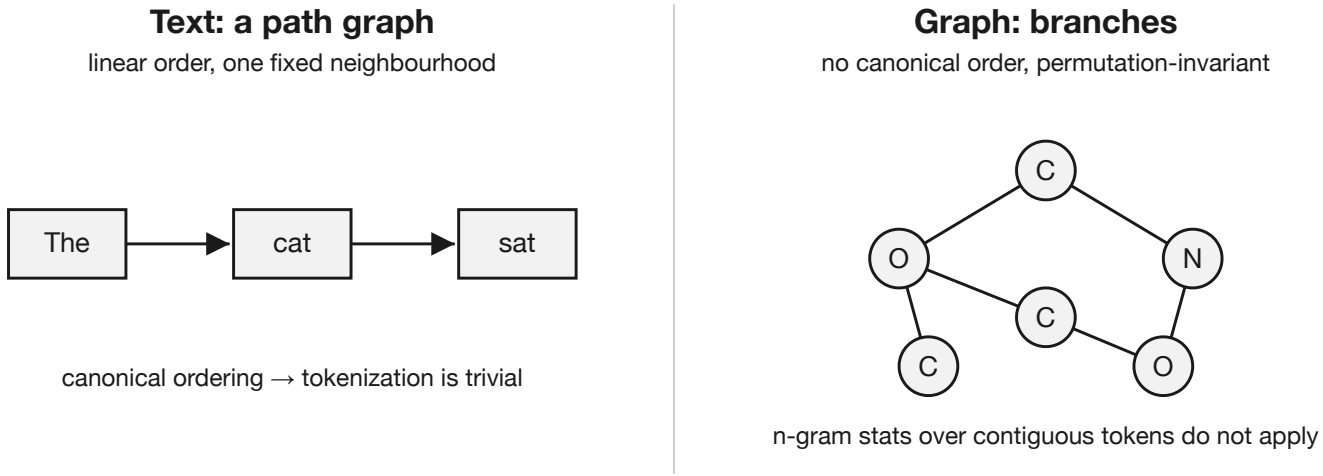

Figure 3: text versus graph. Text is a path graph with one fixed order; a general graph branches and has no canonical ordering.

- **Text is a path graph:** a single linear order and one fixed neighbourhood, so tokenization is straightforward.
- **General graphs break those assumptions:** neighbourhoods **branch**; there is **no canonical order** (a permutation of nodes is the same graph); n-gram statistics over contiguous tokens do not apply.
- **The idea:** first **serialize** the graph into a symbol sequence, then **tokenize it like text** with BPE, provided the serialization is reversible and deterministic.

2.1 Preliminaries: labeled graphs and isomorphism

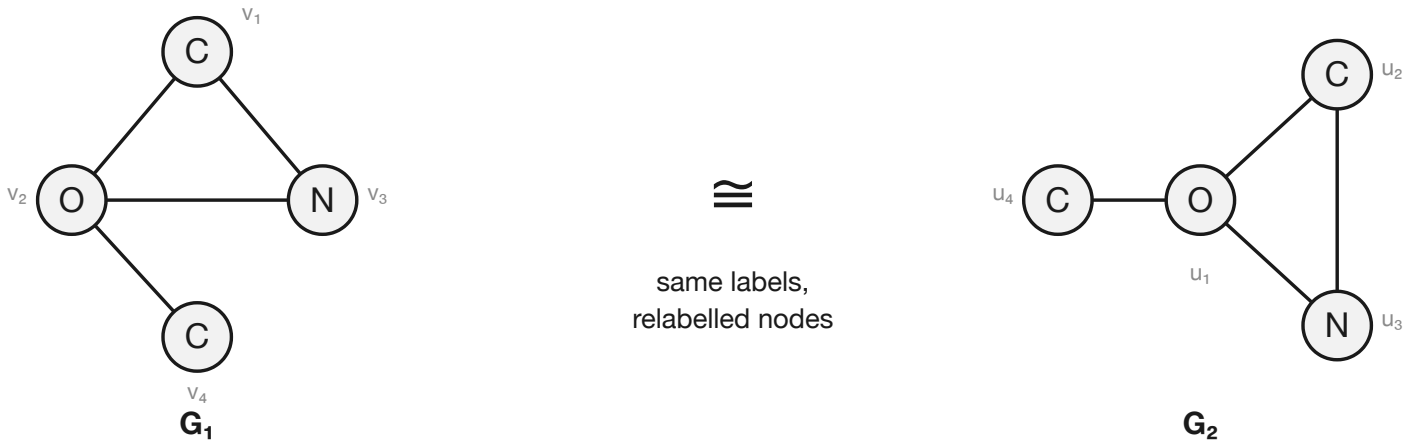

Figure 4: an isomorphism. The same labeled graph drawn two ways; only the node indices differ.

- A **labeled graph** carries a label from an alphabet on every node and edge (for a molecule: atom types, bond types).
- Two graphs are **isomorphic** when one is a relabeling of the other's nodes that preserves all labels and edges: they are the *same* graph.
- A tokenizer must therefore give the **same output for every graph in an isomorphism class**. Node indices must not matter.

2.2 Preliminaries: a serialization must be reversible and deterministic

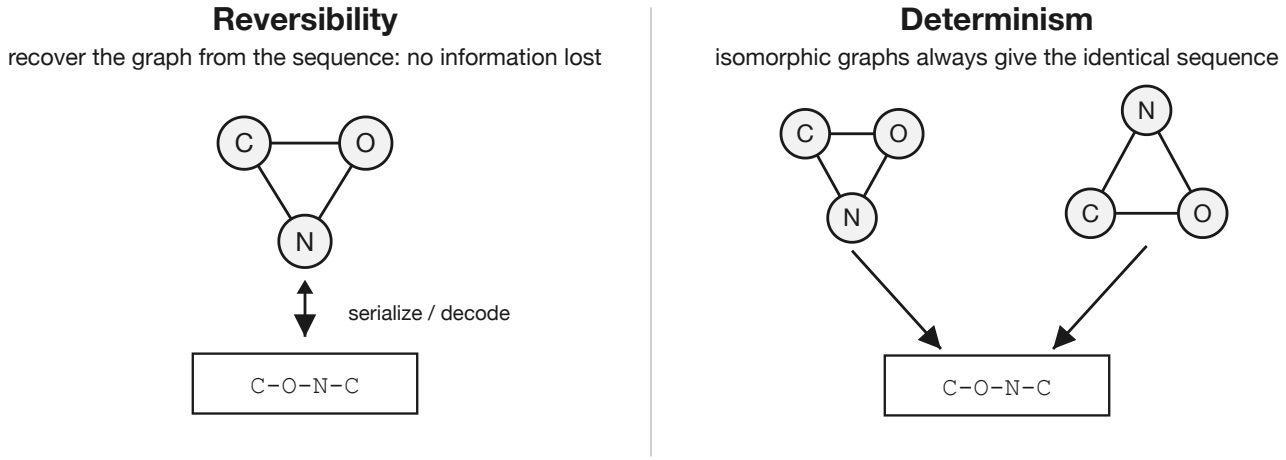

Figure 5: the two properties. Reversibility recovers the graph from the sequence; determinism gives every isomorphic copy the identical sequence.

- **Serialization** maps a graph to a sequence of symbols. To serve as an interface it needs two properties.
- **Reversibility:** the graph can be recovered from the sequence up to isomorphism, so **no information is lost**.
- **Determinism:** the same graph always maps to the same sequence, so the output is **stable across node permutations**. Satisfying **both at once** for general graphs is the core challenge.

2.3 Preliminaries: BPE, how LLM tokenizers learn a vocabulary

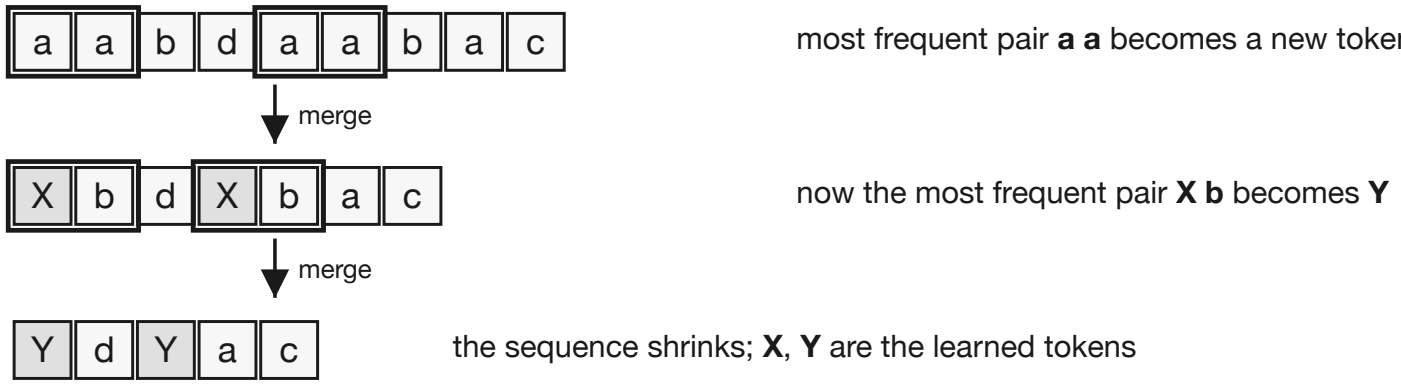

Figure 6: Byte Pair Encoding. Each round merges the most frequent adjacent pair into a new token; the sequence shrinks and the vocabulary grows.

- **Byte Pair Encoding (BPE):** the data-driven tokenizer behind most LLMs. Start from base symbols, then repeatedly **merge the most frequent adjacent pair** into a new token.
- Frequent patterns collapse into **short, meaningful tokens**; rare ones stay split.
- **Precondition:** BPE only finds a unit if it appears as a **frequent, adjacent pair**. That is exactly what the serialization must arrange.

3.1 Method: serialize, then tokenize with BPE

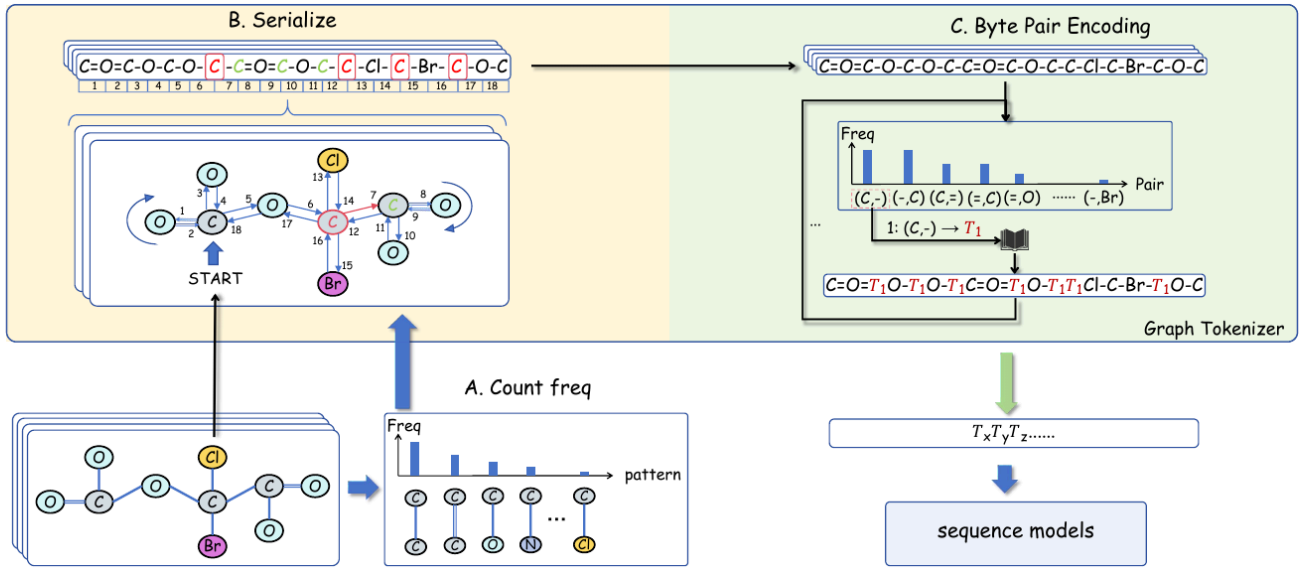

Figure 7: the GraphTokenizer framework. (A) count substructure frequencies over the training graphs; (B) frequency-guided, reversible serialization; (C) BPE learns a vocabulary on the serialized corpus.

- GraphTokenizer composes a serialization f with a BPE tokenizer T: $\Phi = T \circ f$. The structure lives in the **tokenizer**, not the model.
- Output: a token sequence fed to a standard sequence model. **Decode** runs the inverse $f^{-1} \circ T^{-1}$, tokens back to graph.

3.2 Method, step 1: measure what is common in the data

- Before serializing, count small **labeled patterns** so the tokenizer knows which substructures are frequent.
- Basic pattern = a **labeled edge** p = (label of source node, edge, target node). It is the smallest typed relation between two nodes.
- Cheap, permutation-invariant, and stable under isomorphism, which makes it a reliable tie-breaker.

$$C(p) = \sum_{G \in \mathcal{D}} \mathrm{Count}(G, p) \qquad\qquad F(p) = \frac{C(p)}{\sum_{p'} C(p')}$$

- Aggregating over the training set gives the **frequency map** $F(p)$, the data-driven guide for the next step.

3.3 Method, step 2: structure-guided reversible serialization

- Strategy: start from a serialization that is **already reversible**, then **make it deterministic** with the frequency map F.
- **Edge-covering traversals** (Eulerian circuit, Chinese Postman) walk every edge, so the graph can be rebuilt: they are **reversible**. Their only flaw is the **arbitrary choice** at a branch point.
- **The fix:** at each branch, choose the edge whose labeled pattern is **most frequent** under F; fixed lexical rules break any ties. The walk is then fully determined, both **reversible and deterministic** for general graphs.
- **Bonus:** because high-frequency patterns are always preferred, common substructures end up **adjacent in the sequence**, exactly the input BPE needs.

3.4 Method, step 3: learn the vocabulary with BPE

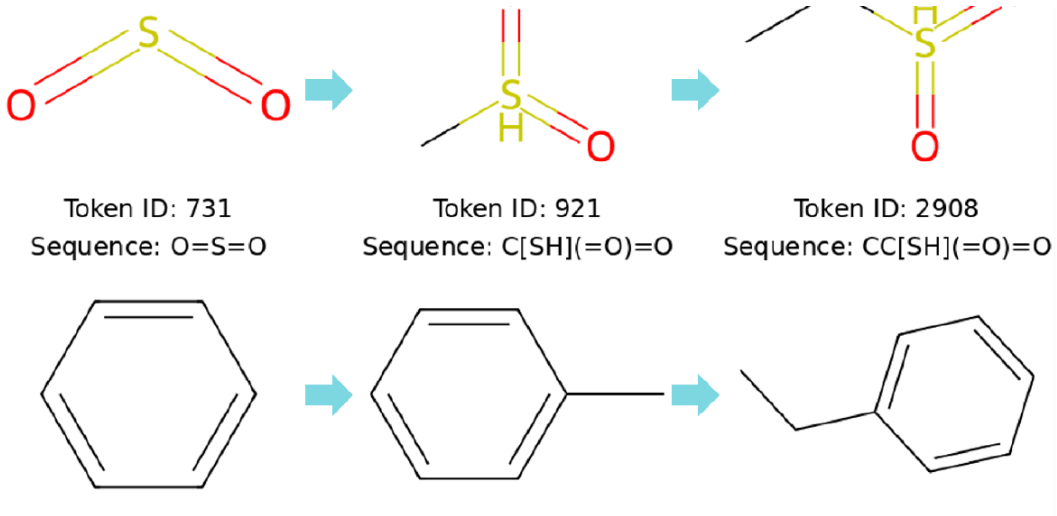

Figure 8: BPE merges on ZINC. Simple substructures (left) are iteratively merged into larger, chemically meaningful tokens: a sulfonyl group, then a benzene ring.

- Run BPE on the serialized corpus: merge the most frequent adjacent pair, then iterate.
- Because serialization already made common substructures **adjacent**, each merge is **not arbitrary compression**; it **discovers a statistically salient subgraph fragment**.
- Every token is a **recoverable subgraph**, so the learned vocabulary is a **structurally informed** representation of the data.

3.5 The result: a reversible bridge into any Transformer

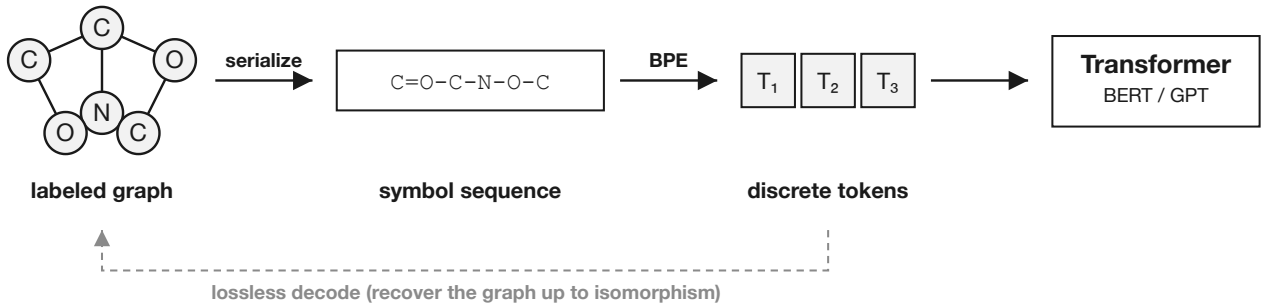

Figure 9: the bridge. Encoding is one preprocessing step and decoding inverts it, so the graph is recovered up to isomorphism while the model stays a standard Transformer.

- The mapping is **bidirectional**: ENCODE (graph to tokens) and DECODE (tokens to graph), so nothing is lost.
- One token sequence feeds any backbone: **BERT** for classification and regression (prepend `[CLS]`, pool), **GPT** for generation, an **LLM** for graph summarization.

> **Graph learning becomes sequence modeling: it inherits long context, efficient attention, and scaling, with no graph-specific architecture.**