I. Supplementary Material

A. The LLM-Guided Subgraph Extraction Prompts

Suggest GNN Features Prompt

-You are an expert in machine learning feature selection, specifically for the GNN graph machine tasks.
-Think about information required to accurately **`<task>`**.
- Return a numbered list of items without explanation.
- Sort the list according to item importance.

Features to BGPs Mapping Prompt

-You are an expert in machine learning feature selection for graph machine learning tasks.
- The following describes the **`<KG>`** knowledge graph schema, detailing the relationships between graph entities in a series of triples, one triple per line:
`<KG-schema>`
-Given the following list of key features, select the matching relations from the previous schema.
`<suggested-features>`
-Think carefully and refine your selected/matching items Return the top **`<K>`** matched schema triples sorted by importance.
-Output only one selected triple per line without any explanation.

BGPs To SPARQL Prompt

-You are an expert SPARQL query writer.
- Given the following triples list from the **`<KG>`** knowledge graph schema, write a SPARQL query to select the **¡VT¿** and its associated information given in the following triples list.
- The triples are directed; make sure to fulfill the direction and relation type.
- The query must return the union of sub-select statements in the form ?s ?p ?o.
- Each triple is Subject Entity - relation - Object Entity.
- Start with the **`<VT>`** node.
`<BGP-List>`
`<SPARQL-Example>`
——————Rules————————
1- write nested select sub-queries and Union them.
2- In single-hop nested select, make sure to start the first BGP with the variable ?s.
3- In tow-hop or more nested select:
3.1 Start the first BGP with the variable ?s, then use other variable names for next BGPs.
3.2 used the last connected entity as the subject, as shown in the previous example.
4- Generate only the SPARQL query without any explanation.
5- Make sure to use each given BGP triple.
6- add the BGP: 'Values ?s ¡VT-List¿.' to the end of each sub query.
7- Refine all rules and the query syntax. 8- Do invent new relations i.e, dblp:authoredBy can not be dblp:Authored, But you can start with ?o instead of ?s.
Example:
?s a dblp:Publication.
?s dblp:authoredBy ?o.
——— Should Be ———
?o a dblp:author. ?s dblp:authoredBy ?o.

SPARQL Refine Prompt

```
-You are an expert SPARQL query writer. Given the following SPARQL query, re-write it to follow the following rules.
- Rule1: Keep the nested selects and their Union statements.
- Rule2: restructure the n-hop sub-select to choose the latest BGP subject and object and as the select items.
- Example: {?s a prefix:x. ?s prefix:y ?y. ?y prefix:z ?z.}
the latest BGP is ?y prefix:z ?z, then the select items must be: 1- ?y as ?s. 2- ?p. 3- ?z as ?o
——— SPARQL Query ——————-
<sparql-query>
-Refine The Rules and Examples Carefully.
-Return only the Query; do not return any explanation.
¡Answer¿
```

An example of LLM-Guided SPARQL Query generated for the DBLP-PV NC task.

```
PREFIX dblp: <https://dblp.org/rdf/schema>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
 SELECT ?s ?p ?o  FROM <https://www.dblp.org>
WHERE { { SELECT ?s ?p ?o WHERE { ?s a dblp:Publication. ?s dblp:title ?o. BIND( ”dblp:title” AS ?p). VALUES ?s {<VT-List>}. }} UNION
{ SELECT ?s ?p ?o WHERE { ?s a dblp:Publication. ?s dblp:yearOfEvent ?o. BIND( ”dblp:yearOfEvent” AS ?p). VALUES ?s {<VT-List>}. }} UNION
{ SELECT ?s ?p ?o WHERE { ?s a dblp:Publication. ?s dblp:publishedInJournalVolume ?o. BIND( ”dblp:publishedInJournalVolume” AS ?p). VALUES ?s {<VT-List>}. }} UNION
{ SELECT ?author ?p ?o WHERE { ?s a dblp:Publication. ?s dblp:authoredBy ?author. ?author dblp:primaryAffiliation ?o. BIND( ”dblp:primaryAffiliation” AS ?p). VALUES ?s {<VT-List>}. }} UNION
{ SELECT ?author ?p ?o WHERE { ?s a dblp:Publication. ?s dblp:authoredBy ?author. ?author rdfs:label ?o. BIND( ”rdfs:label” AS ?p). VALUES ?s {<VT-List>}. }} UNION
{ SELECT ?s ?p ?o WHERE { ?s a dblp:Publication. ?s dblp:publishedInSeries ?o. BIND( ”dblp:publishedInSeries” AS ?p). VALUES ?s {<VT-List>}. }} UNION
{ SELECT ?s ?p ?o WHERE { ?s a dblp:Publication. ?s dblp:numberOfCreators ?o. BIND( ”dblp:numberOfCreators” AS ?p). VALUES ?s {<VT-List>}. }} UNION
{ SELECT ?s ?p ?o WHERE { ?s a dblp:Publication. ?s dblp:publishedBy ?o. BIND( ”dblp:publishedBy” AS ?p). VALUES ?s {<VT-List>}. }} UNION
{ SELECT ?s ?p ?o WHERE { ?s a dblp:Publication. ?s dblp:doi ?o. BIND( ”dblp:doi” AS ?p). VALUES ?s {<VT-List>}. }} }
```

B. *KG-WISE with varying inference query size*

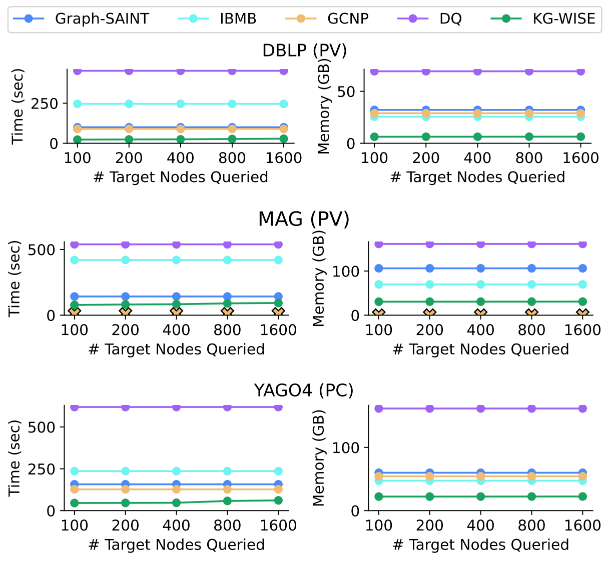

Fig. 1. Node classification inference time and memory usage of KG-WISE compared to SOTA GNN accelerators as the number of target nodes per query increases. The top, middle, and bottom rows correspond to the DBLP, MAG, and YAGO NC tasks, respectively. KG-WISE exhibits superior performance in all cases.

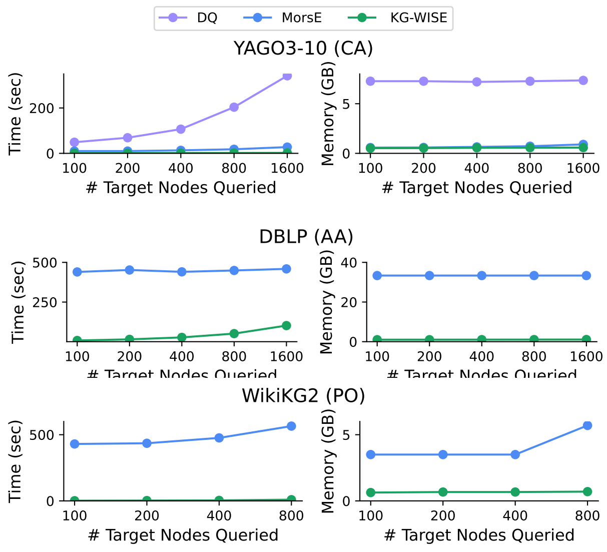

Fig. 2. Link prediction inference time and memory usage of KG-WISE vs. SOTA GNN accelerators across varying numbers of target nodes per query. The top, middle, and bottom rows correspond to the YAGO3-10-CA, DBLP-PV, and WikiKG-PO LP tasks, respectively. Baseline methods show near-linear inference time and constant memory usage, regardless of query size. In contrast, KG-WISE exhibits sub-linear growth in both time and memory, enabling efficient, query-aware, on-demand inference.

TABLE I

CO_2 EMISSIONS AND ENERGY CONSUMPTION OF KG-WISE AND GRAPH-SAINT. KG-WISE GENERATES 60% LESS CO_2 EMISSIONS AND CONSUMES 62% LESS ENERGY.

Metric	GraphSAINT	KG-WISE
Total Energy (Wh)	1.19	0.42
Energy per Target Node (Wh)	1.2×10^{-3}	0.4×10^{-3}
Total Emission (g CO_2)	1.5×10^{-1}	0.53×10^{-1}
Emission per Target Node (g CO_2)	1.5×10^{-4}	0.53×10^{-4}
Relative CO_2 Efficiency	-	2.83x

C. *KG-WISE Energy and Carbon-Emission*

Training and inference for large-scale GNN models require substantial computational resources.Consequently, measuring the carbon footprint of these models has become essential. We compare the carbon emissions during inference of Graph-SAINT and KG-WISE, leveraging the CodeCarbon V2.8.3 [**?**] software package for measurement. Carbon dioxide (CO_2) emissions are computed using the expression $C \times E$, where C is the carbon intensity of electricity in grams g and E is the energy consumed by the computational infrastructure in Watts. The value of C depends on the energy production methods specific to each country, which may involve varying proportions of renewable energy, natural gas, and petroleum. CodeCarbon maintains a database tracking the carbon intensity values for different countries.

To measure E, the tool tracks the power consumption of three key components: CPU, GPU, and memory. Since GPUs are not used in our setup, they are excluded. Dynamic CPU power utilization is derived from Intel's RAPL files when available; otherwise, the tool uses the static Total Draw Power (TDP) specification of the CPU. Memory energy consumption is estimated at 0.375W/GB. We use the tool's 'Process Tracking' mode to measure the memory utilized by the process itself rather than the entire system memory.

For this experiment, we perform inference on 1K nodes from the DBLP node classification task using both Graph-SAINT and KG-WISE. Table I shows that Graph-SAINT consumes a total energy of 1.19 Wh, while KG-WISE consumes only 0.42 Wh.

These measurements also enable us to calculate the energy usage per target node, which is significantly lower for KG-WISE. Similarly, the CO_2 emissions are 1.5×10^{-1}g and 0.53×10^{-1}g for Graph-SAINT and KG-WISE, respectively.

Overall, the proposed KG-WISE reduces both energy consumption and carbon emissions by 2.83 times compared to Graph-SAINT. This improvement is primarily due to more efficient model loading and reduced forward-pass complexity. These significant reductions highlight the potential of KG-WISE to improve task performance while promoting environmentally sustainable AI practices.