CGP-Tuning: Structure-Aware Soft Prompt Tuning for Code Vulnerability Detection

Presenter: Ruijun Feng

01/22/2025

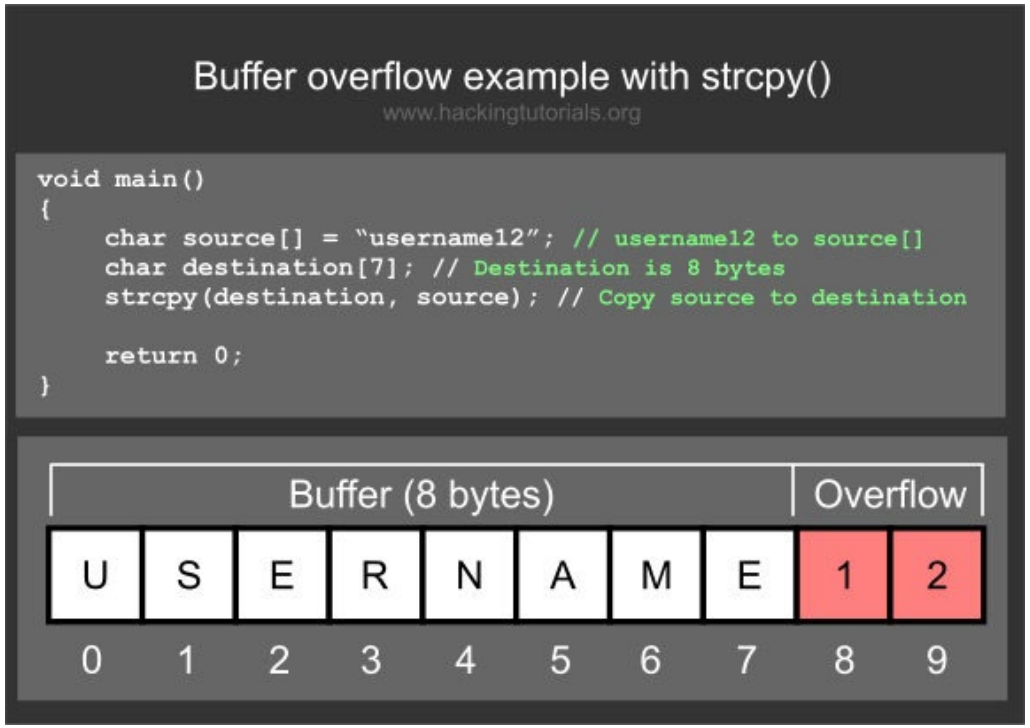
Contents

1. Literature Review
2. Our Work
3. Future Work

Literature Review

What is Vulnerability Detection

- Vulnerability detection is about identifying **potential vulnerabilities** in a **given code snippet**.
  - E.g., buffer overflow, the stored data exceeds the range of the array.



What is Vulnerability Detection

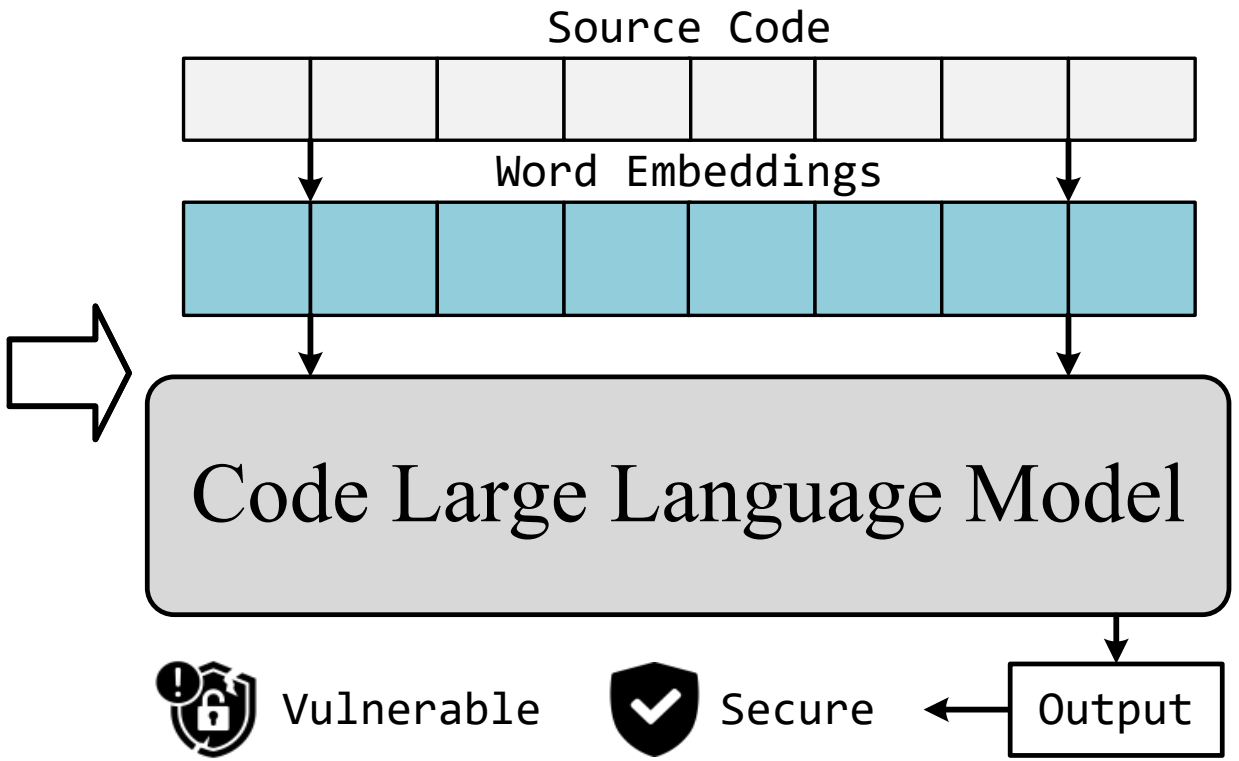
- Vulnerability detection is about identifying **potential vulnerabilities** in a **given code snippet**.
- It is an important problem to ensure software reliability.

Aim of the Research

- Use **code large language model** to **improve vulnerability detection**, making it **more precise and efficient**.

Source Code

```
① int main() {
②   char buffer[10];
③   char input[50];
④   fgets(input, sizeof(input), stdin);
⑤   strcpy(buffer, input);
⑥   return 0;
}
```



What is Code Large Language Model

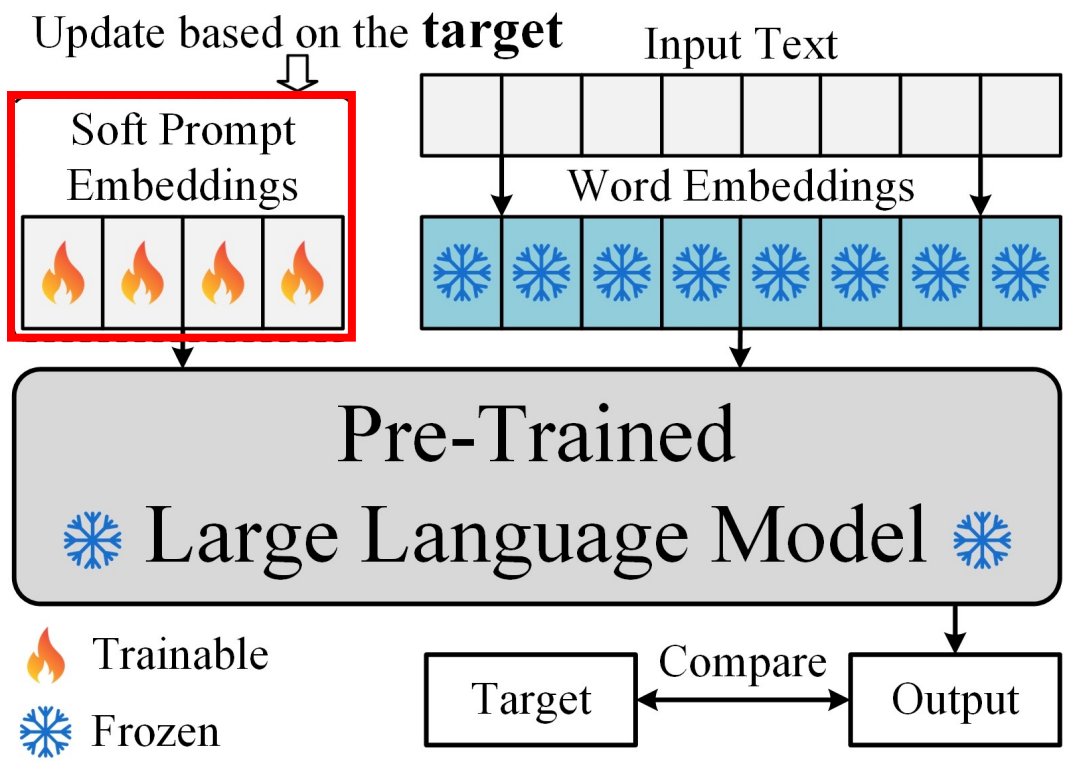
- Code large language models (LLM) are specialized LLMs **pre-trained** on **source code and related documentation** to capture **general** coding patterns.

What is Code Large Language Model

- Code large language models (LLM) are specialized LLMs **pre-trained** on **source code and related documentation** to capture **general** coding patterns.
- Pre-trained code LLMs lack **task-specific knowledge**, so they are typically **fine-tuned on the target dataset** to enable them better perform various code-related tasks.

Soft Prompt Tuning

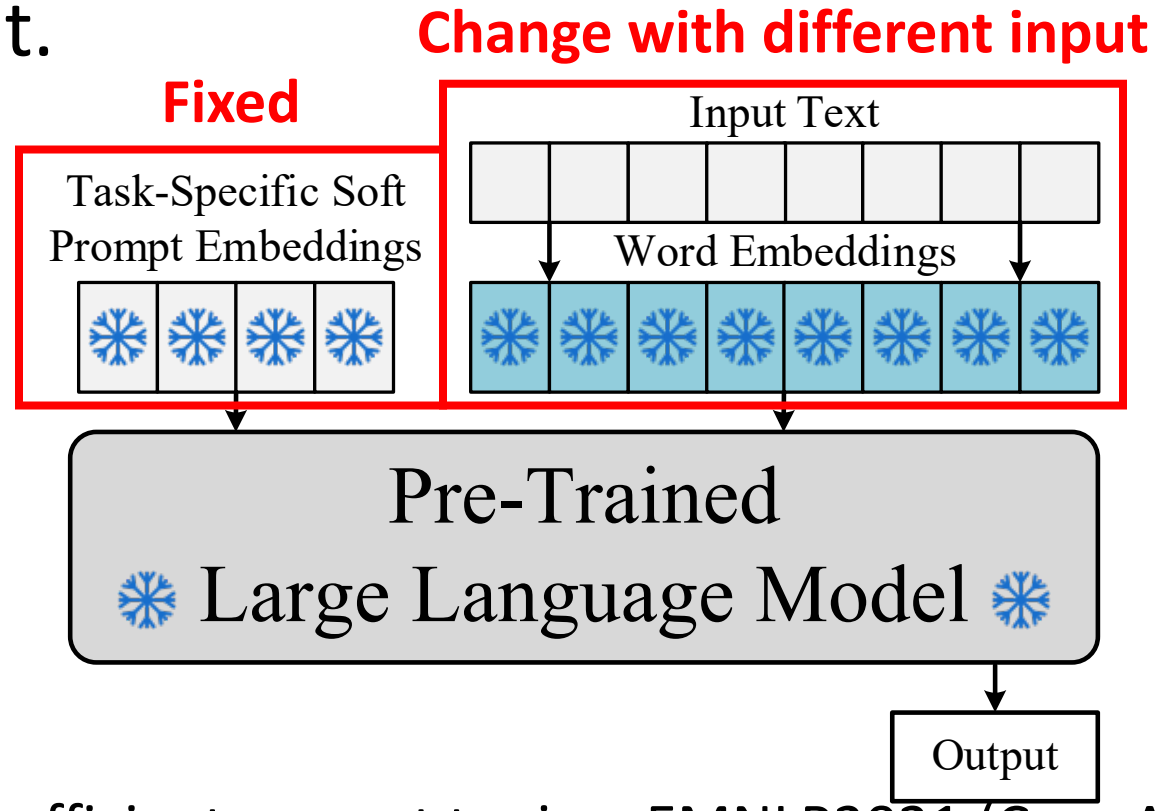
- During **fine-tuning**, the pre-trained LLM and the word embeddings are **frozen**, only the **soft prompt embeddings** are updated based on target.



Lester et al. The power of scale for parameter-efficient prompt tuning. EMNLP2021 (Core A*).

Soft Prompt Tuning

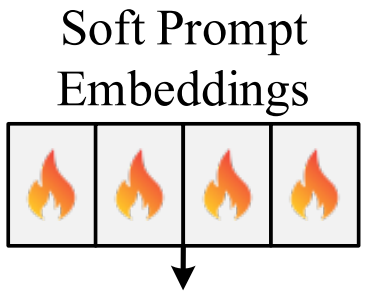
- During **inference**, the soft prompt embeddings are **fixed** and serve as the prompt to instruct pre-trained LLM to **perform certain task** for different input.

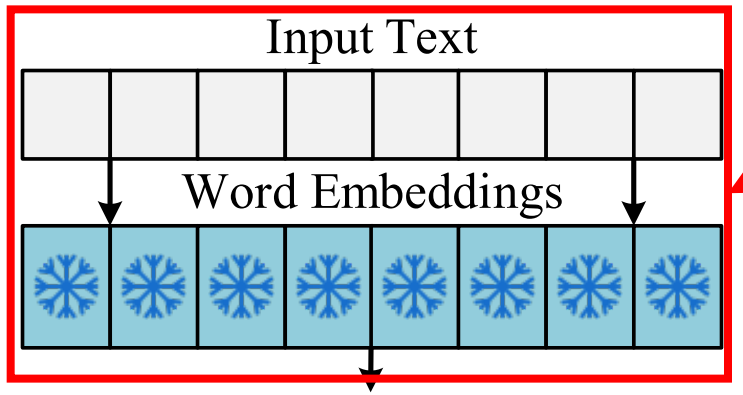


Lester et al. The power of scale for parameter-efficient prompt tuning. EMNLP2021 (Core A*).

Limitations of Soft Prompt Tuning

- Code LLM treat source code as **plain text sequences**, it ignores the **structural information** within the source code, like control flow.





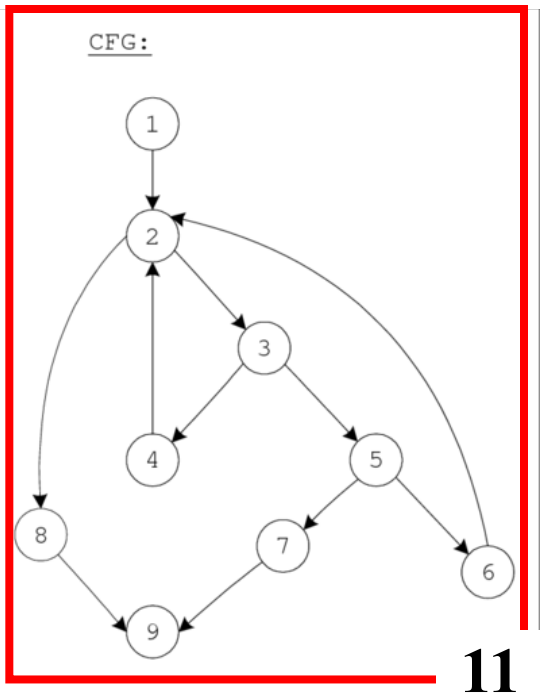
control flow ✗

missing

Source Program:

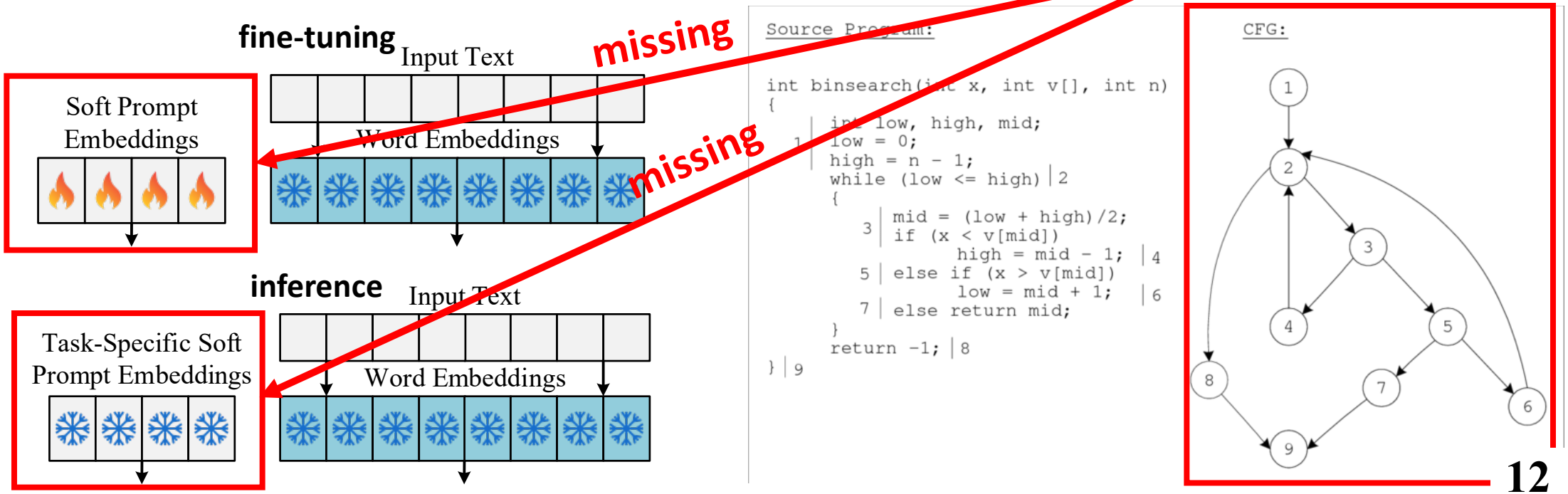
```
int binsearch(int x, int v[], int n)
{
    int low, high, mid;
1   low = 0;
    high = n - 1;
    while (low <= high) |2
    {
3       mid = (low + high)/2;
        if (x < v[mid])
            high = mid - 1;  |4
5       else if (x > v[mid])
            low = mid + 1;   |6
7       else return mid;
    }
    return -1; |8
} |9
```

CFG:



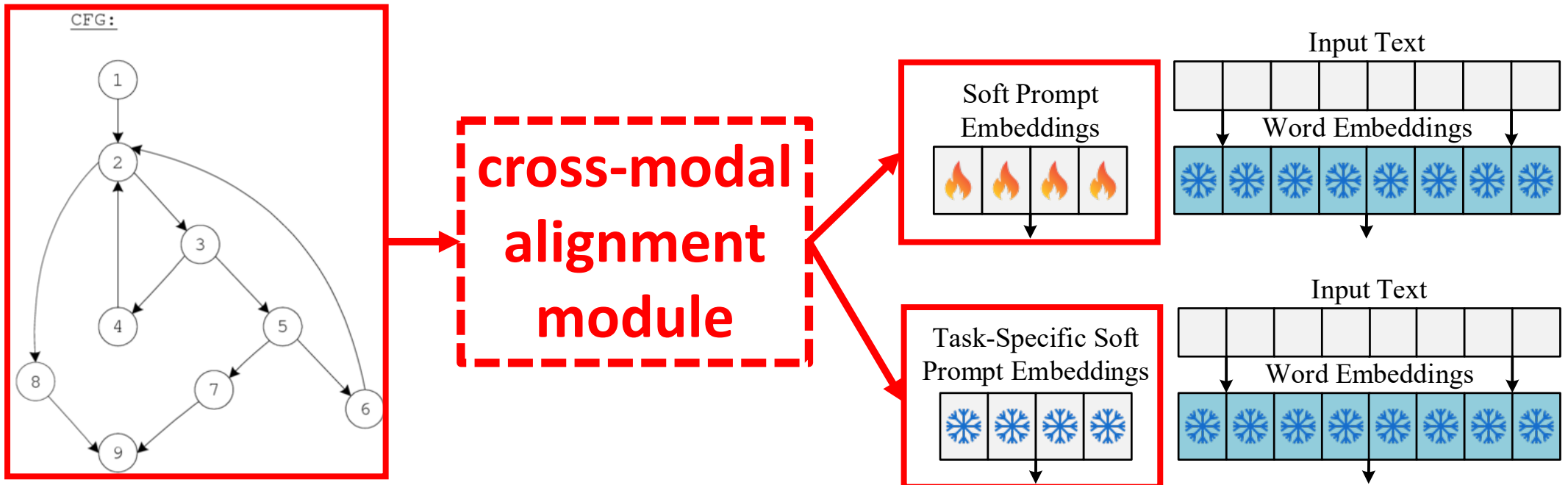
Limitations of Soft Prompt Tuning

- The learned soft prompt embeddings **cannot** encode the **dynamic** structural information of different input during both **fine-tuning** and **inference**.



Graph-Enhanced Soft Prompt Tuning

Use **cross-modal alignment module** to encode **dynamic graph-based structural information** into the soft prompt embeddings during both **fine-tuning and inference**, making the process **structure-aware**.



structural information

Graph-Enhanced Soft Prompt Tuning

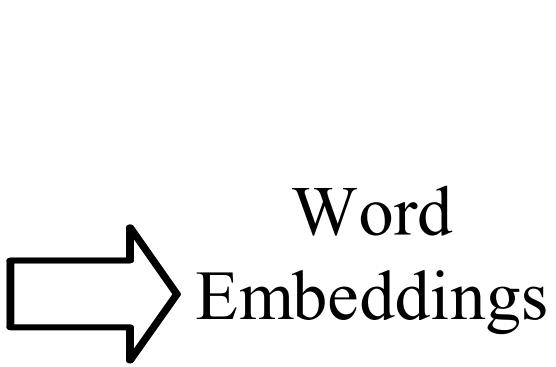
- Projector-based Methods
- Cross-attention-based Methods

Preliminaries

- Given a piece of source code, it can be represented as word embeddings through the dictionary and embedding layer of LLM as **text features**.

Source Code

```
① int main() {
②   char buffer[10];
③   char input[50];
④   fgets(input, sizeof(input), stdin);
⑤   strcpy(buffer, input);
⑥   return 0;
}
```



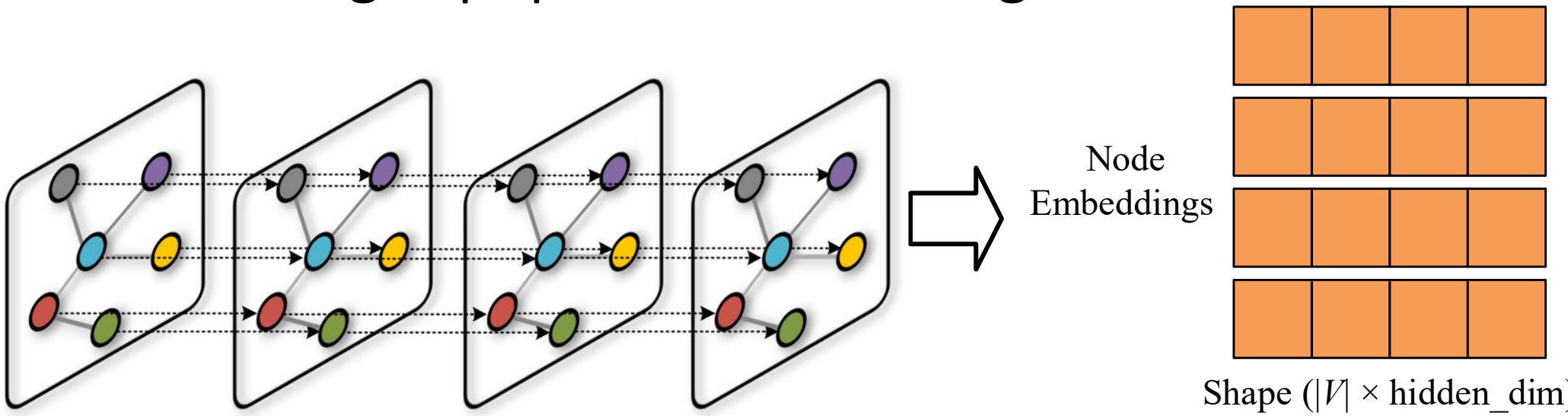
Shape ($N \times$ hidden_dim)

Preliminaries

- Suppose the graph representation of source code is denoted as $G = (V, E)$, where V is the set of nodes, and E is the set of edges.

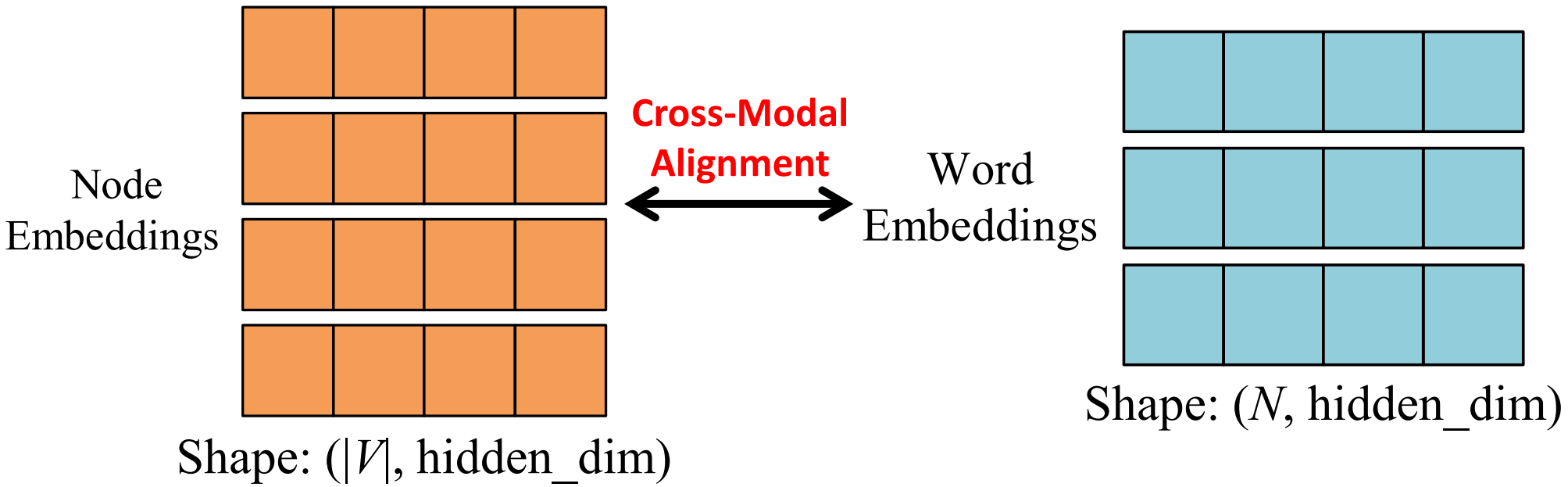
Preliminaries

- Suppose the graph representation of source code is denoted as $G = (V, E)$, where V is the set of nodes, and E is the set of edges.
- Graph neural network is used to extract **graph features** from G and get $|V|$ node embeddings.

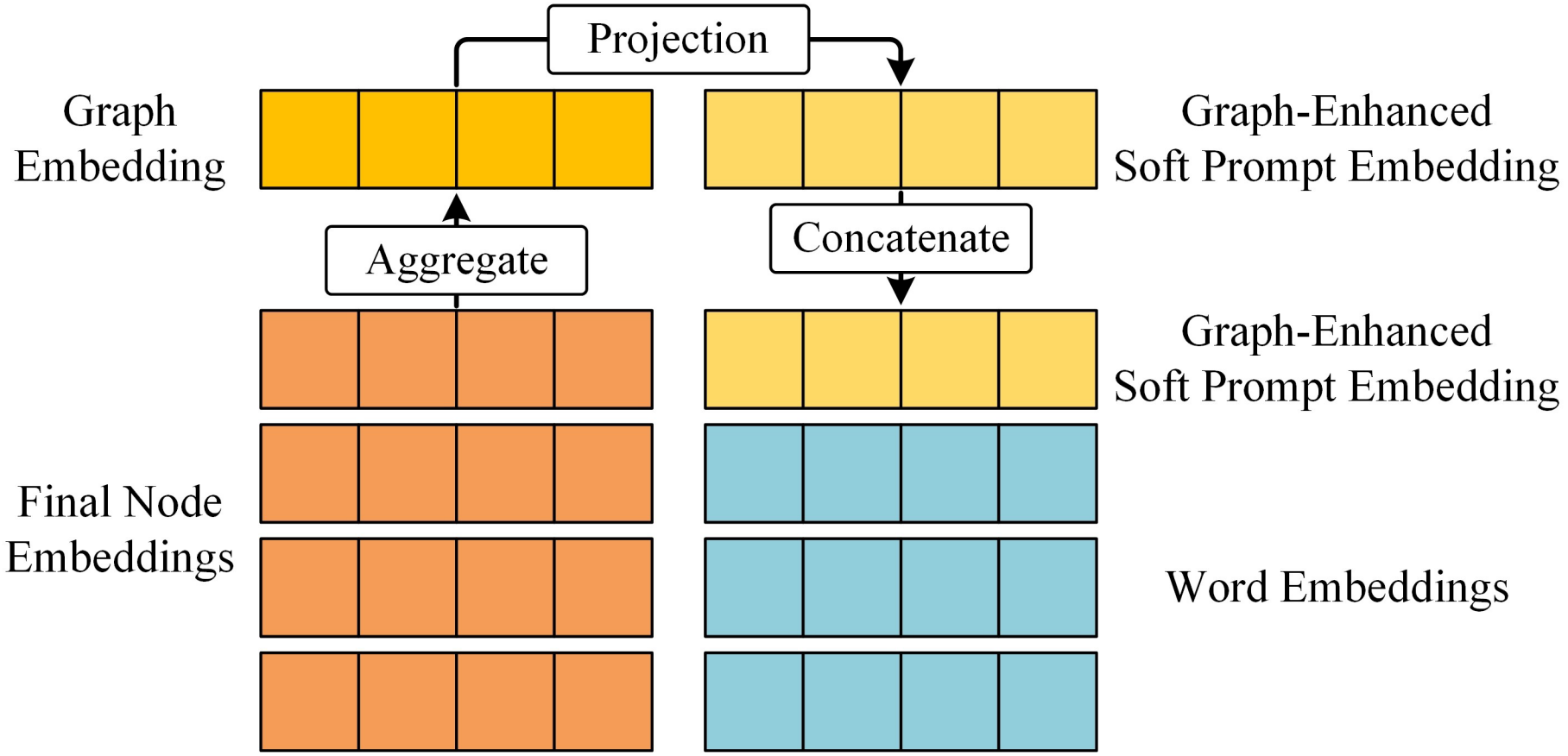


Preliminaries

- The key to graph-enhanced soft prompt tuning is to perform **cross-modal alignment** between **graph features** ($|V|$ node embeddings) and **text features** (N word embeddings).

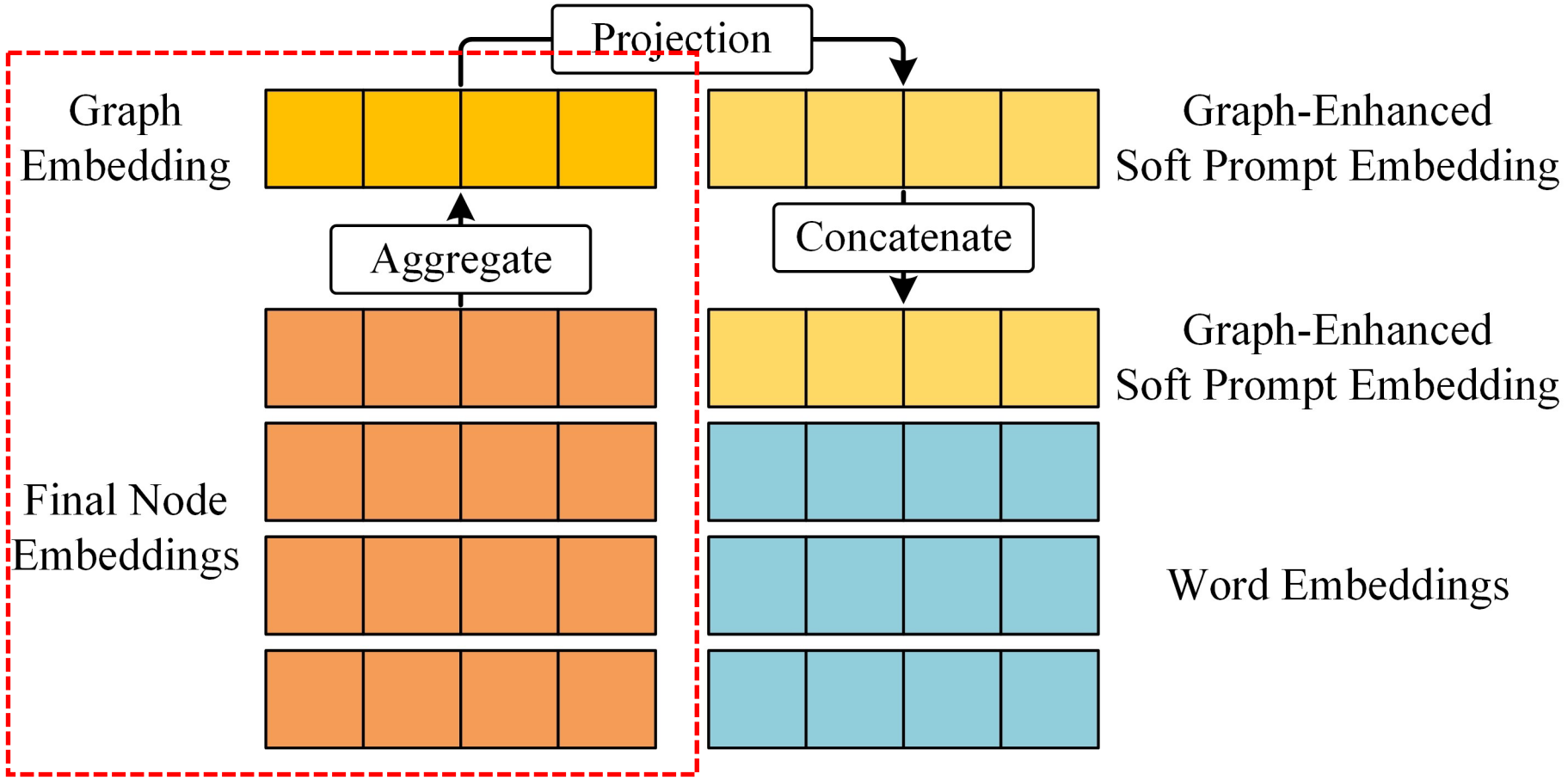


Projector-based Method



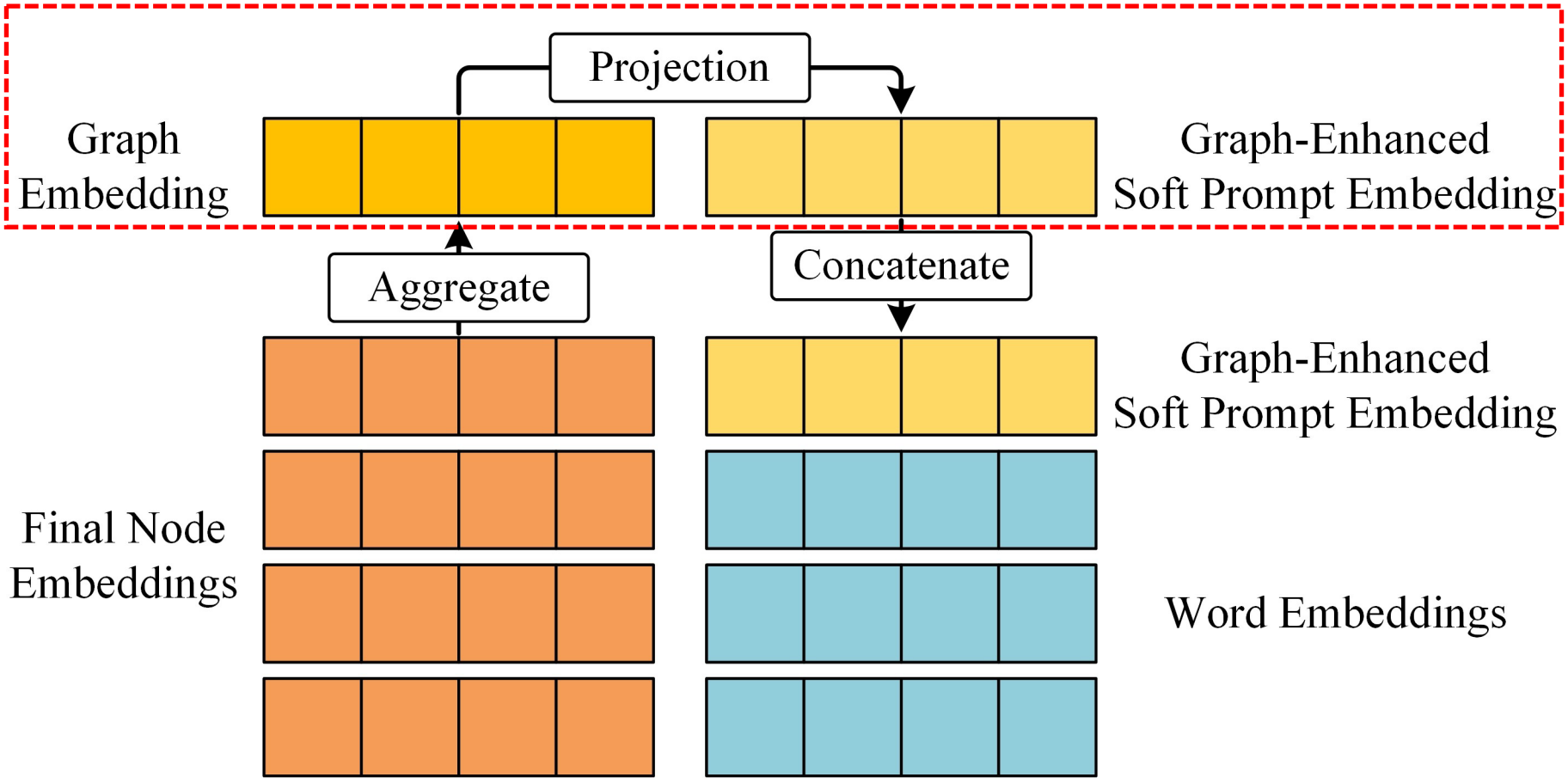
He et al. G-Retriever: Retrieval-Augmented Generation for Textual Graph Understanding and Question Answering. NeurIPS2024 (Core A*).

Projector-based Method



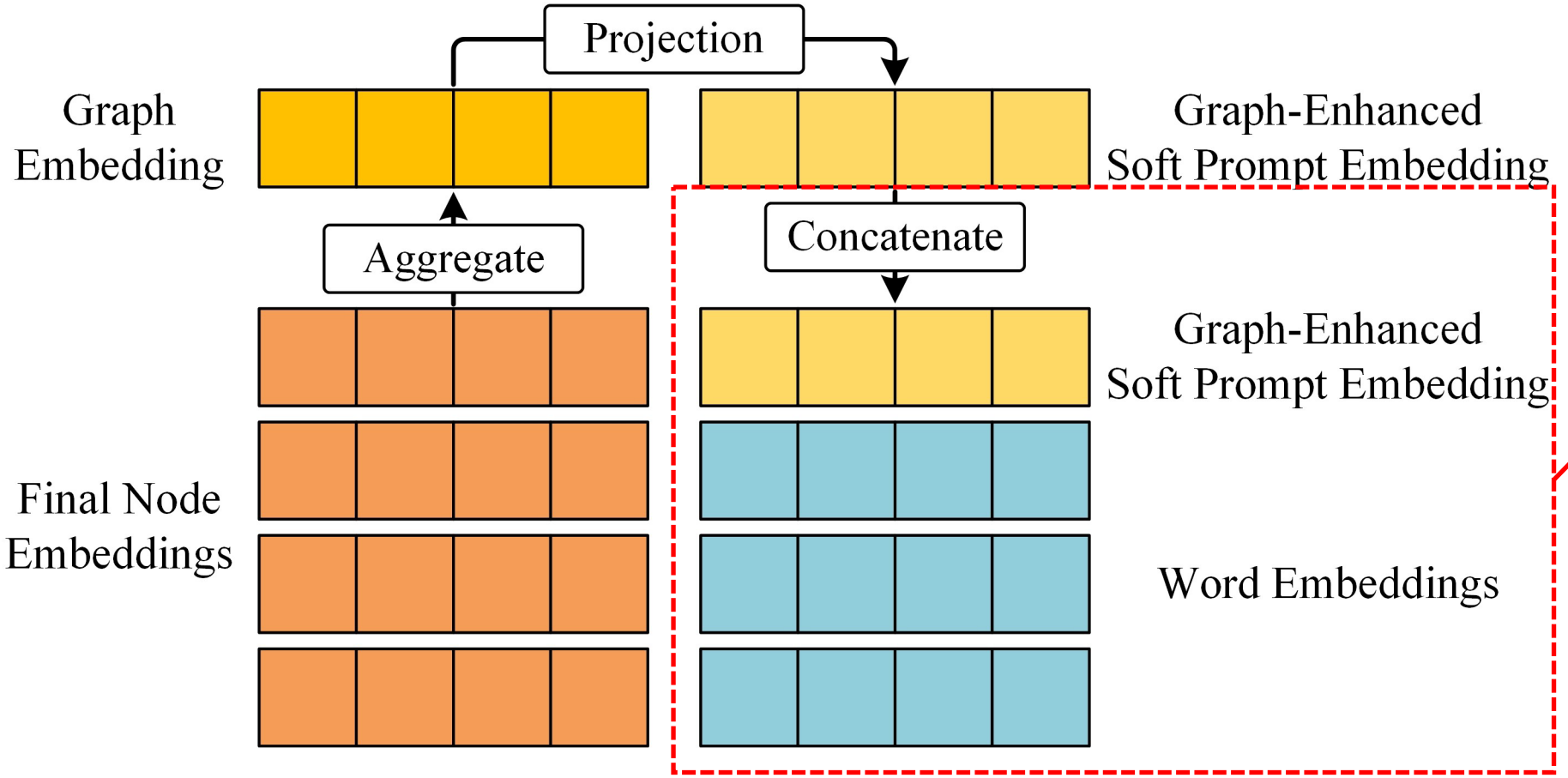
He et al. G-Retriever: Retrieval-Augmented Generation for Textual Graph Understanding and Question Answering. NeurIPS2024 (Core A*).

Projector-based Method



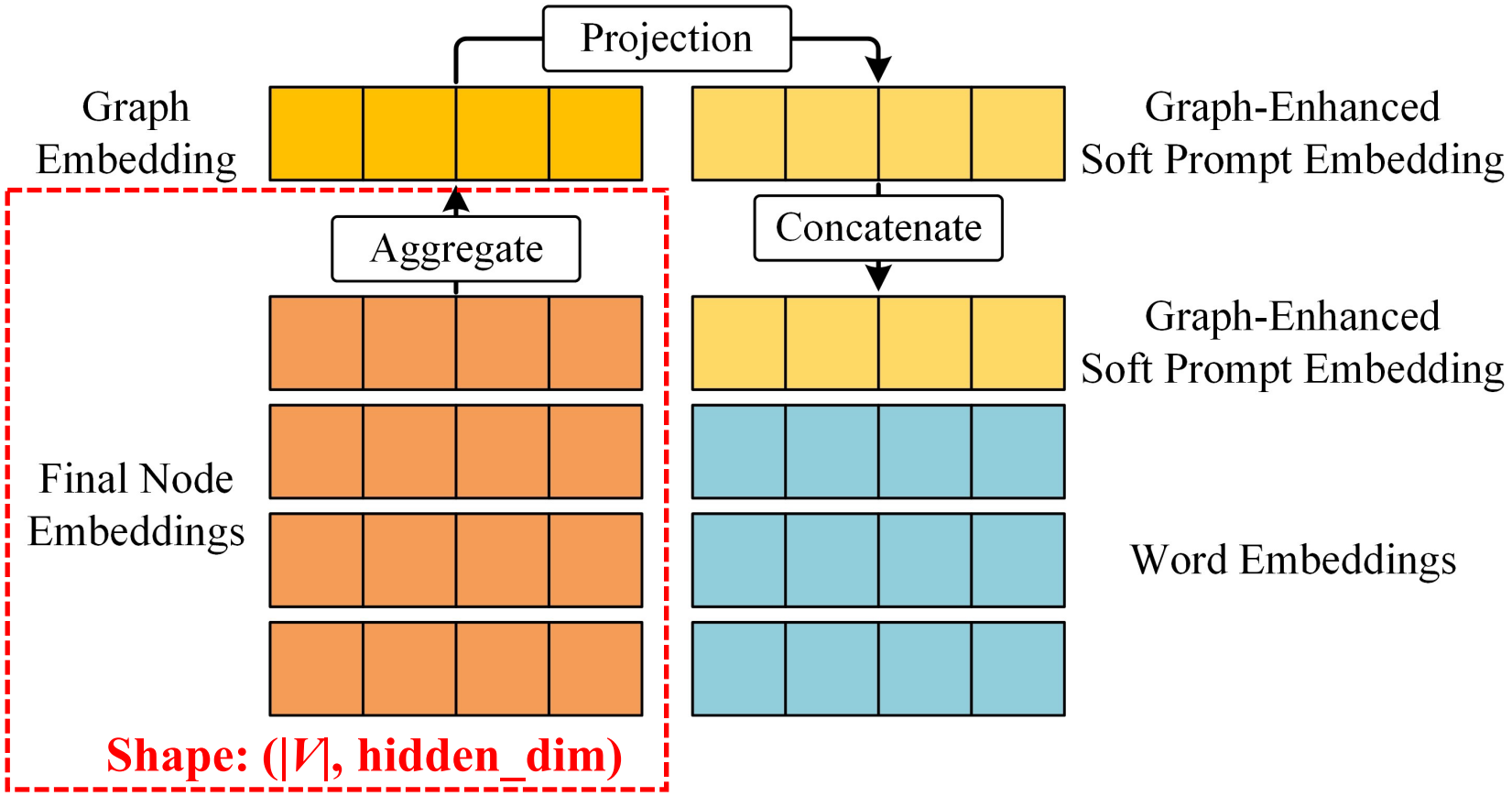
He et al. G-Retriever: Retrieval-Augmented Generation for Textual Graph Understanding and Question Answering. NeurIPS2024 (Core A*).

Projector-based Method



He et al. G-Retriever: Retrieval-Augmented Generation for Textual Graph Understanding and Question Answering. NeurIPS2024 (Core A*).

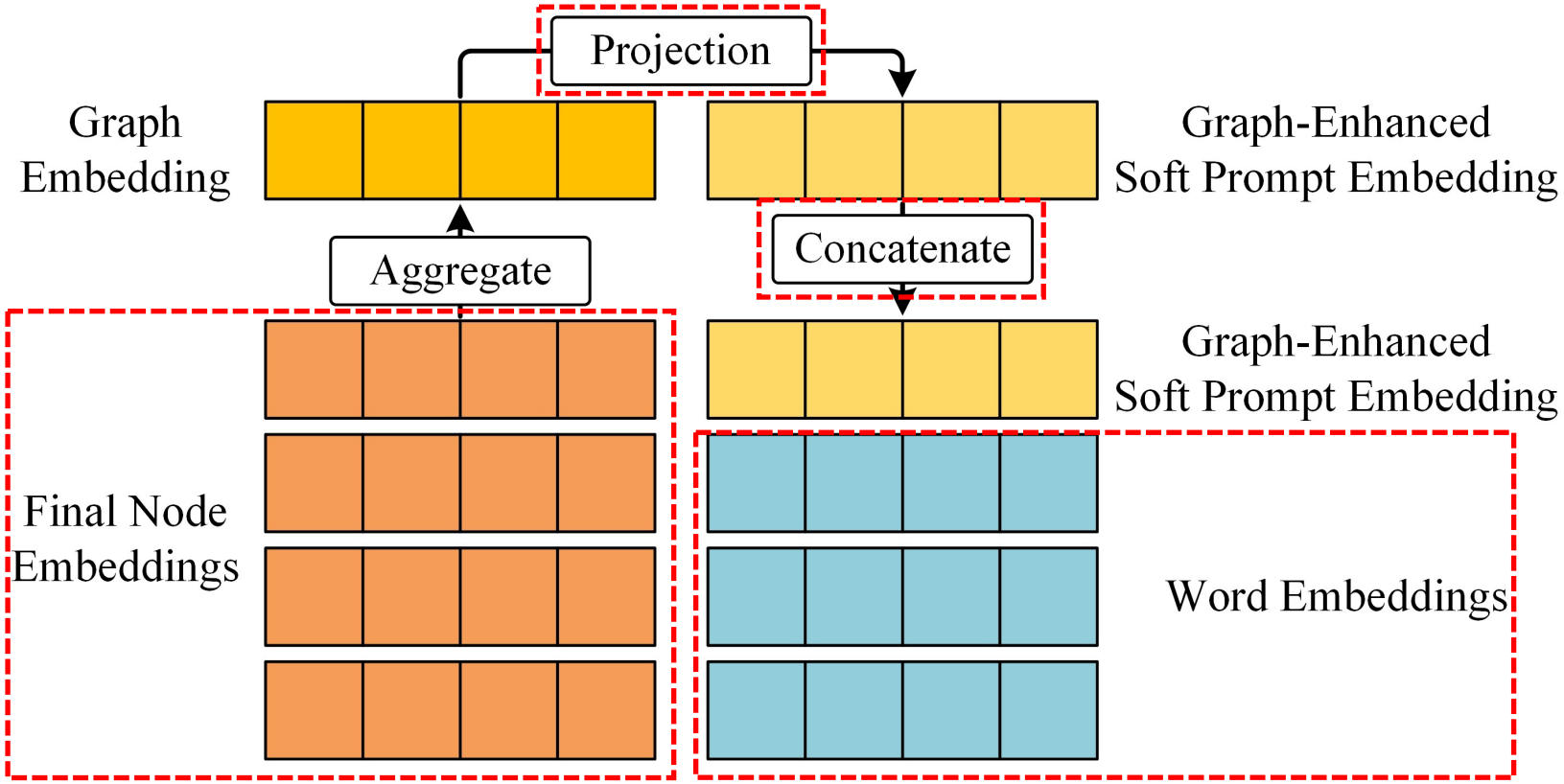
Projector-based Method



Computational load proportional to $|V|$

He et al. G-Retriever: Retrieval-Augmented Generation for Textual Graph Understanding and Question Answering. NeurIPS2024 (Core A*).

Projector-based Method

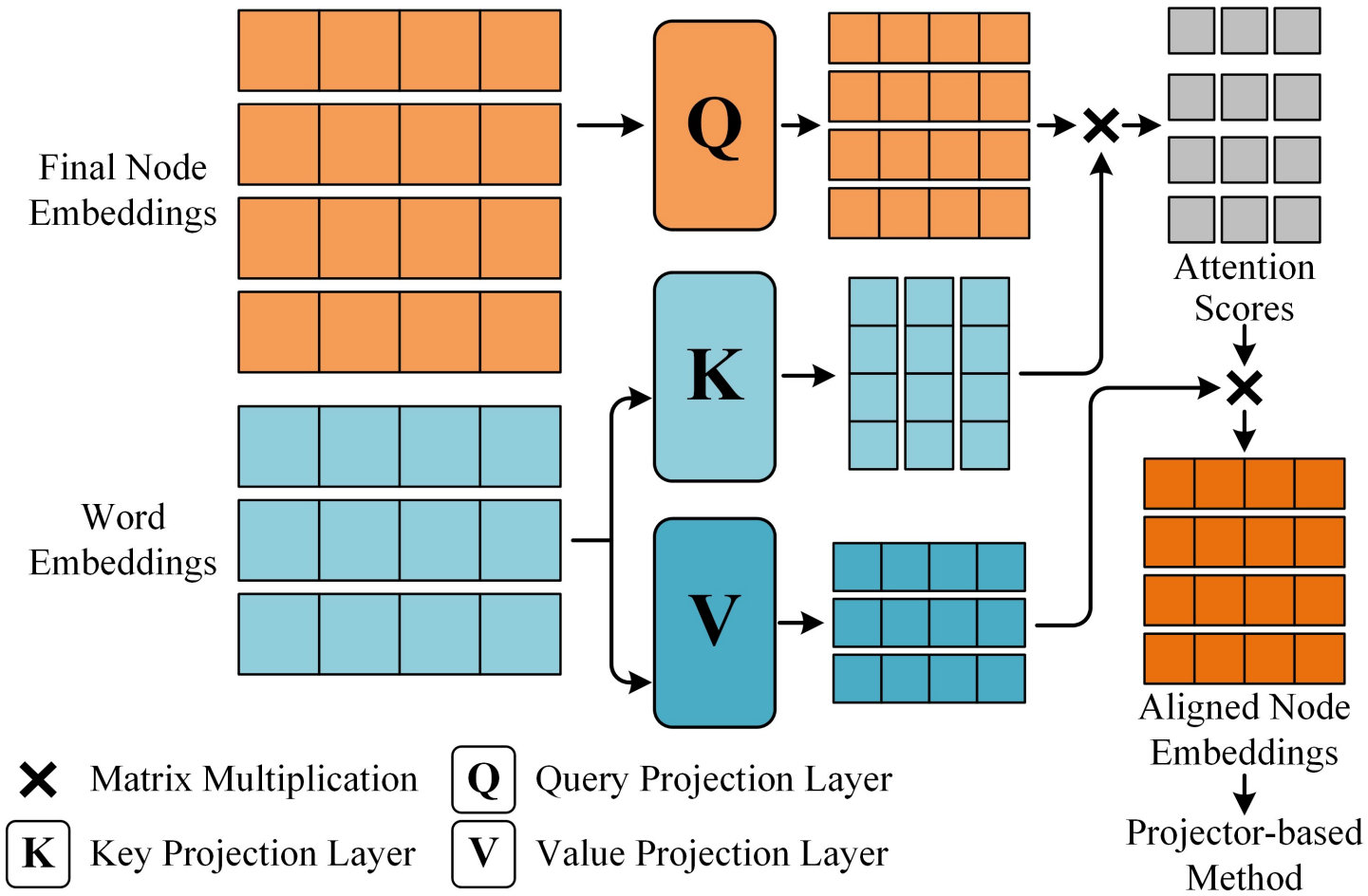


Computational load proportional to $|V|$

Graph-text interactions ✗

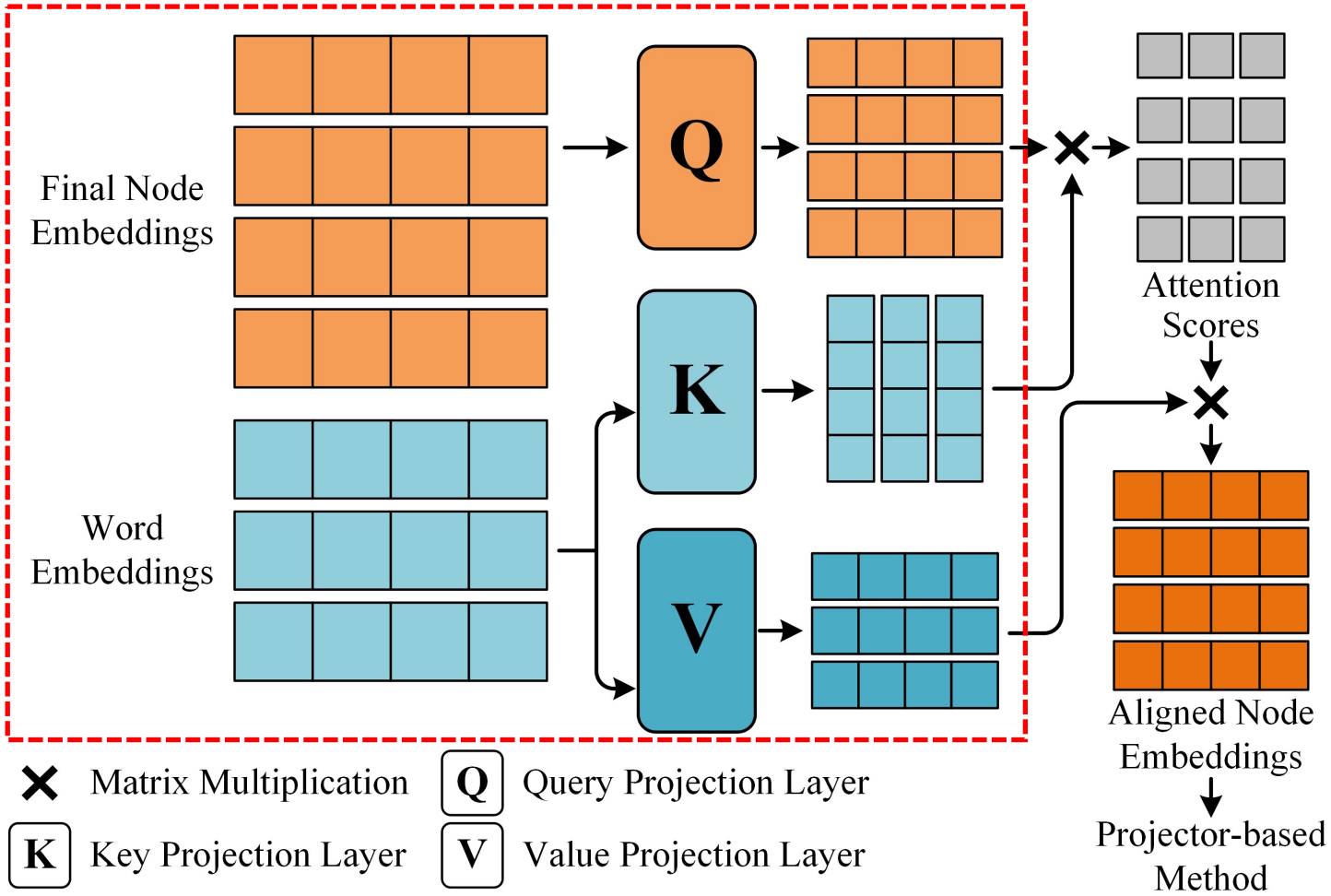
He et al. G-Retriever: Retrieval-Augmented Generation for Textual Graph Understanding and Question Answering. NeurIPS2024 (Core A*).

Cross-attention-based Method



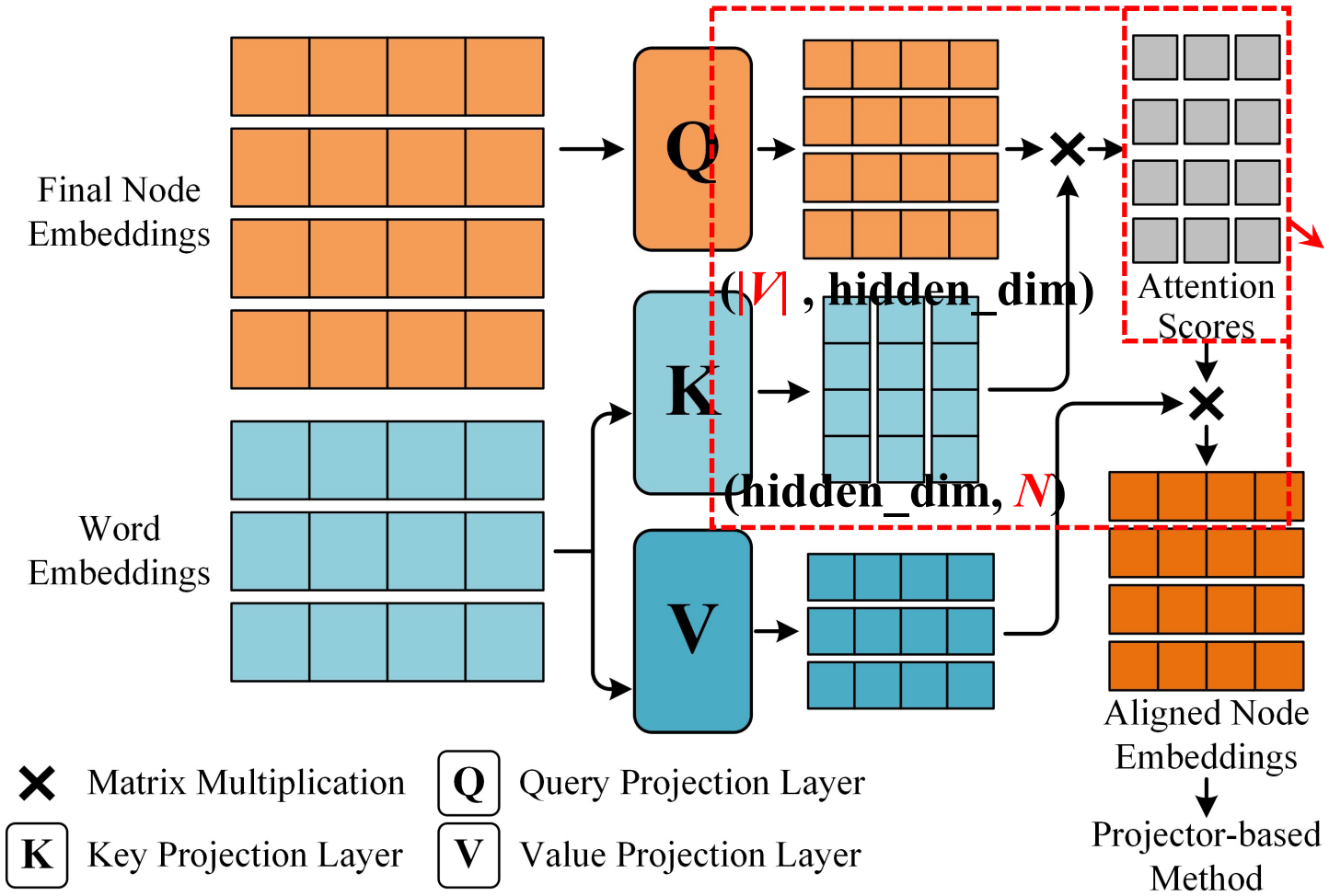
Tian et al. Graph Neural Prompting with Large Language Models. AAAI2024 (Core A*).

Cross-attention-based Method



Tian et al. Graph Neural Prompting with Large Language Models. AAAI2024 (Core A*).

Cross-attention-based Method

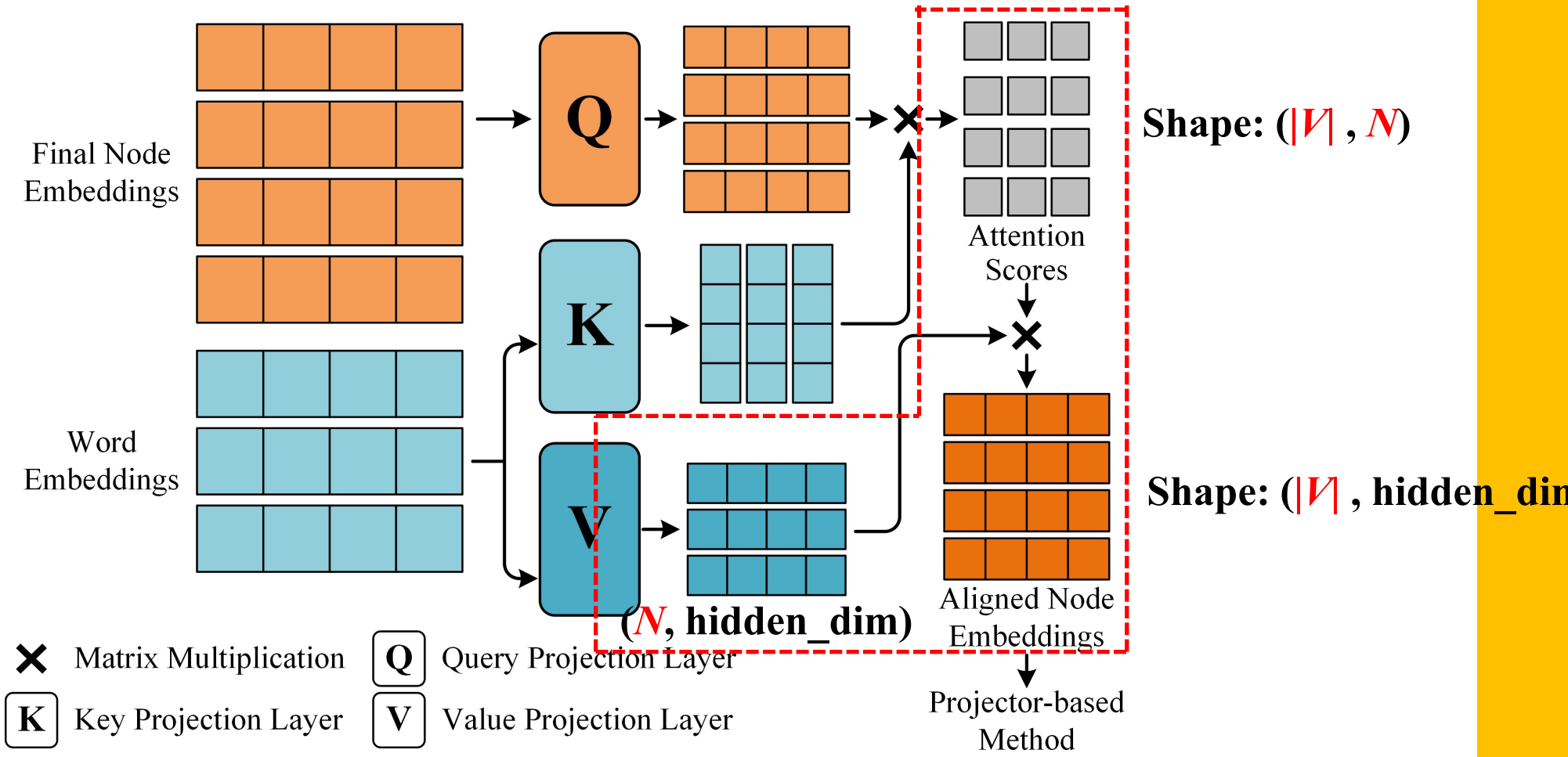


Shape: $(|V|, N)$

Quantitatively assess the relevance between each **node** and **word**.

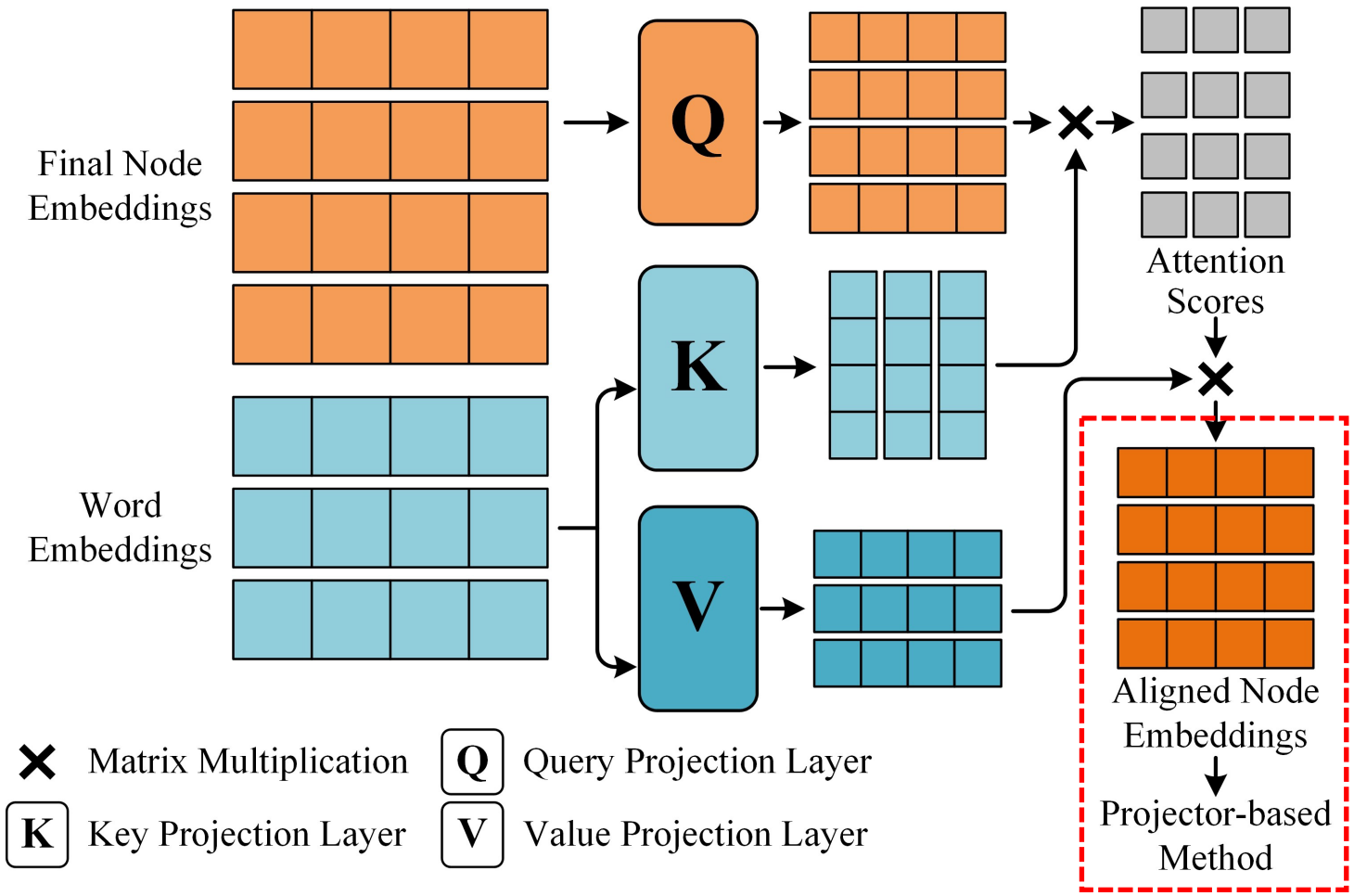
Tian et al. Graph Neural Prompting with Large Language Models. AAAI2024 (Core A*).

Cross-attention-based Method



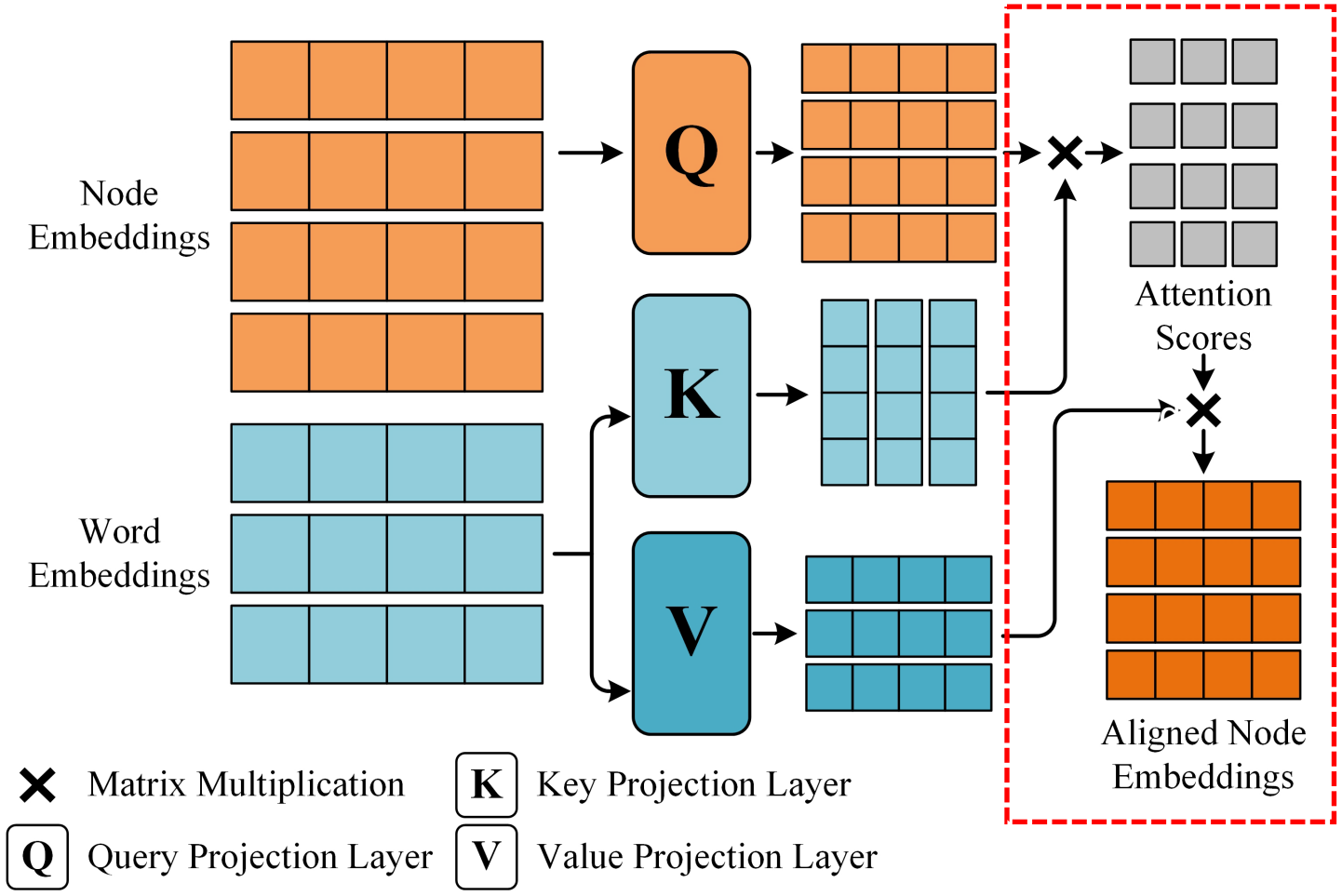
Tian et al. Graph Neural Prompting with Large Language Models. AAAI2024 (Core A*).

Cross-attention-based Method



Tian et al. Graph Neural Prompting with Large Language Models. AAAI2024 (Core A*).

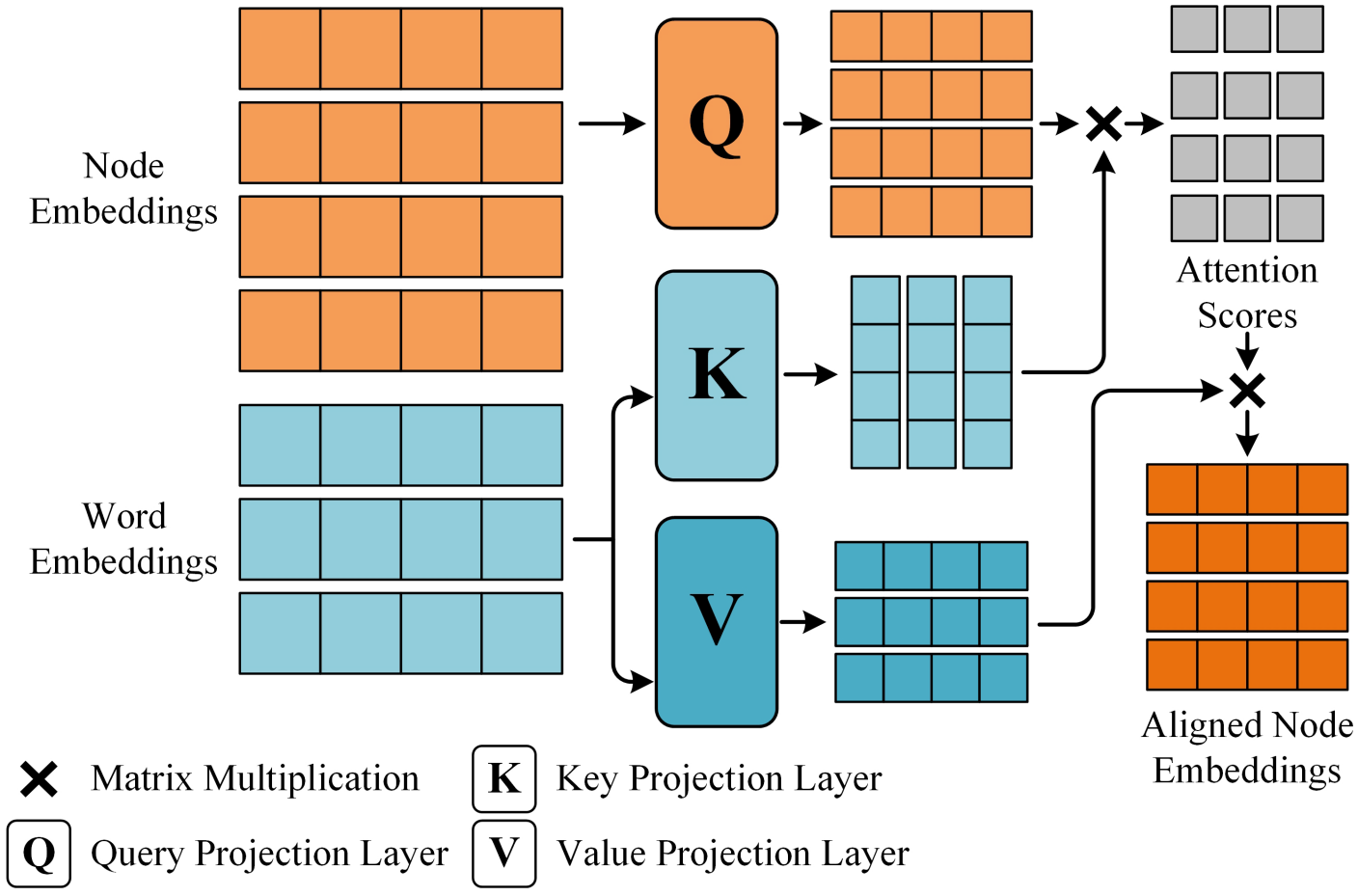
Cross-attention-based Method



Graph-text interactions ✓

Tian et al. Graph Neural Prompting with Large Language Models. AAAI2024 (Core A*).

Cross-attention-based Method



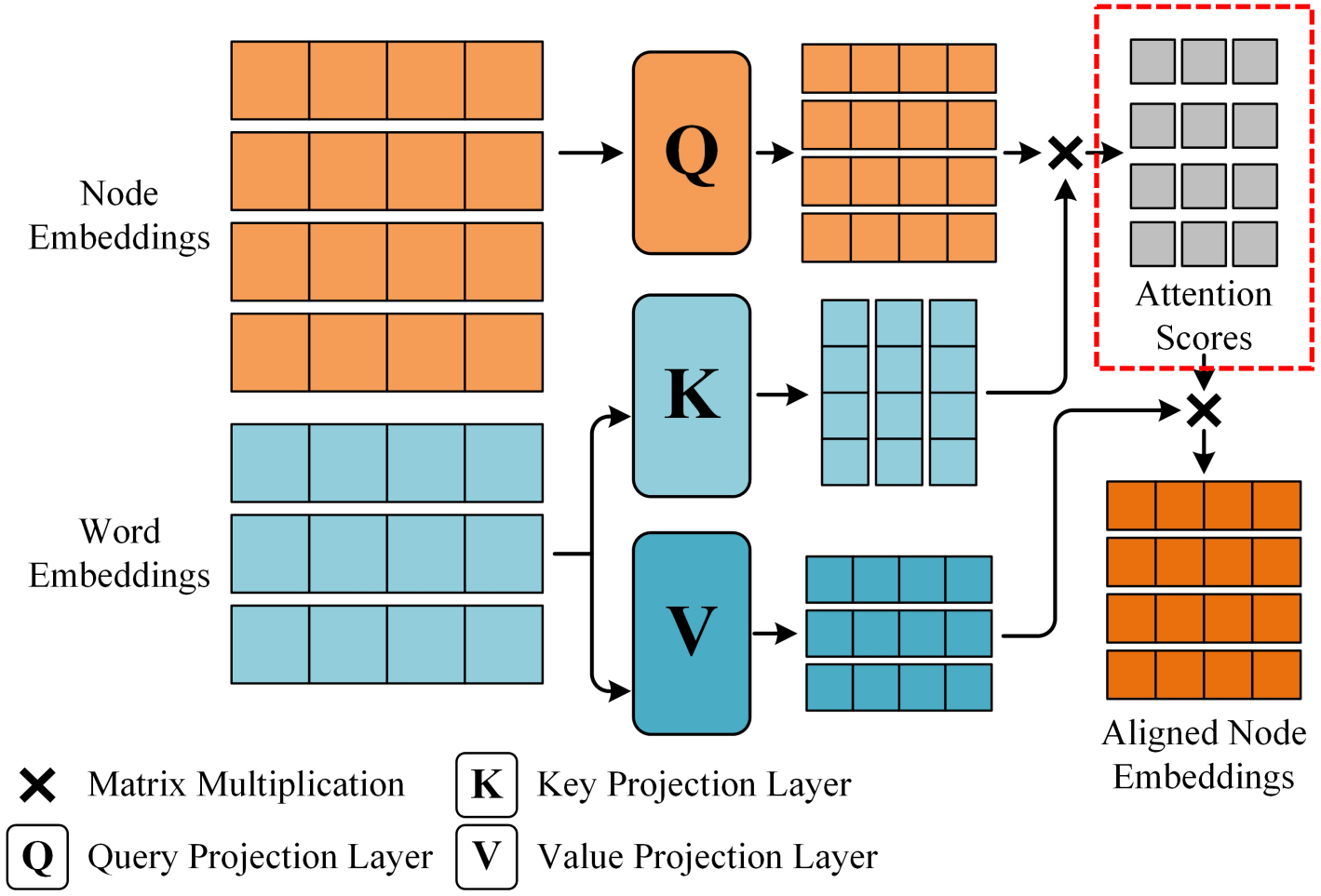
Graph-text interactions ✓

Computational load proportional to : $|V|N+|V|$

Not scalable for code with large graphs.

Tian et al. Graph Neural Prompting with Large Language Models. AAAI2024 (Core A*).

Cross-attention-based Method



A piece of source code with **83K** nodes and **64K** tokens. Under **32-bit float** precision, attention scores matrix will consume **19G** of memory.

Tian et al. Graph Neural Prompting with Large Language Models. AAAI2024 (Core A*).

Limitations of Graph-Enhanced Soft Prompt Tuning

- Limitation 1: They don’t provide good trade-offs between **computational efficiency** and **comprehensiveness**.

Limitations of Graph-Enhanced Soft Prompt Tuning

- Limitation 1: They don't provide good trade-offs between **computational efficiency** and **comprehensiveness**.
  - Either the computational efficiency is good but does not account for graph-text interactions (projector-based method), or it accounts for it but with low computational efficiency (cross-attention-based method).

Limitations of Graph-Enhanced Soft Prompt Tuning

- Limitation 1: They don't provide good trade-offs between **computational efficiency** and **comprehensiveness**.

- Limitation 2: They are designed for **general graph-related** task (e.g., biological network analysis).

Limitations of Graph-Enhanced Soft Prompt Tuning

- Limitation 1: They don't provide good trade-offs between **computational efficiency** and **comprehensiveness**.

- Limitation 2: They are designed for **general graph-related** task (e.g., biological network analysis).
  - They cannot fully explore the rich semantics (**e.g., node and edge types**) in the graph representation of source code.

Our Work

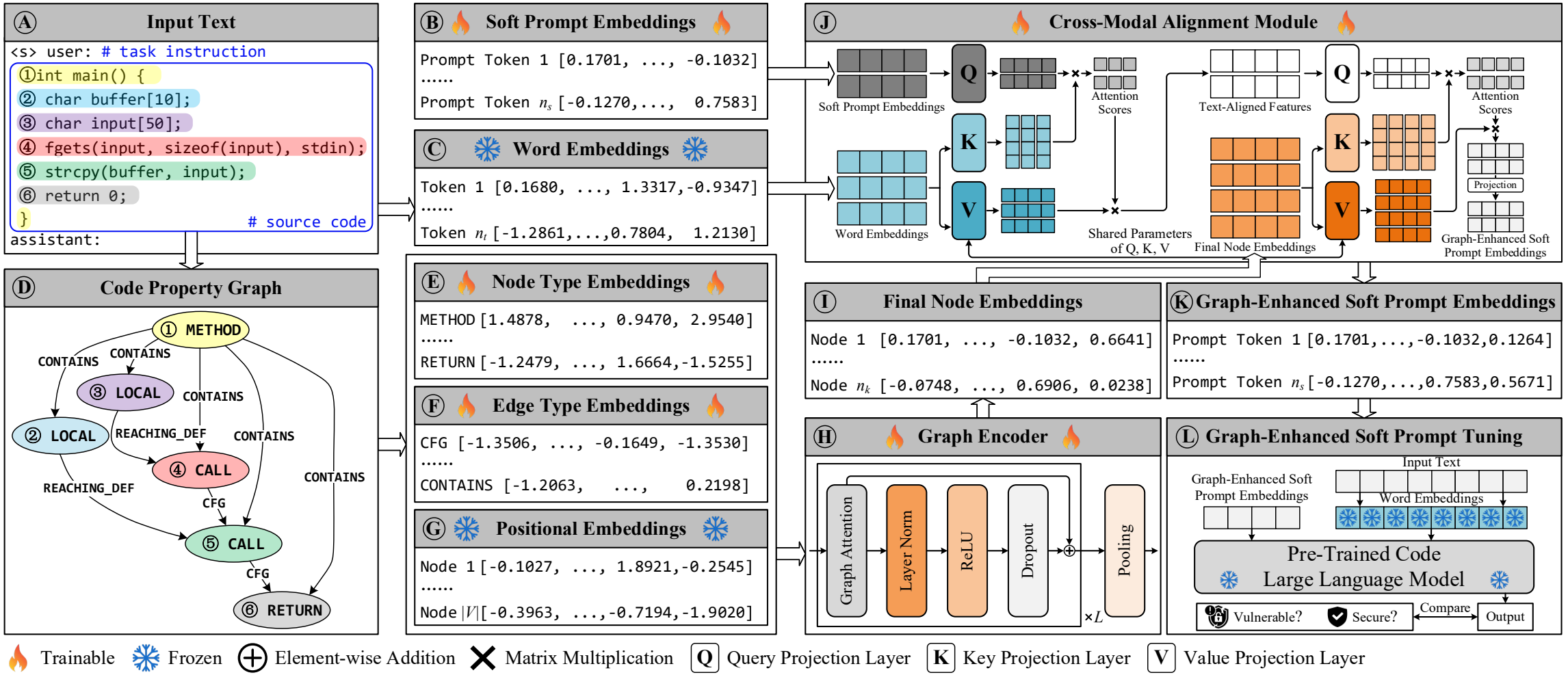
Our Contributions

- Solution to Limitation 1:
  - **Efficient cross-modal alignment module** with a **low computational load** ($|V| + N$) while considering the **graph-text interactions**.

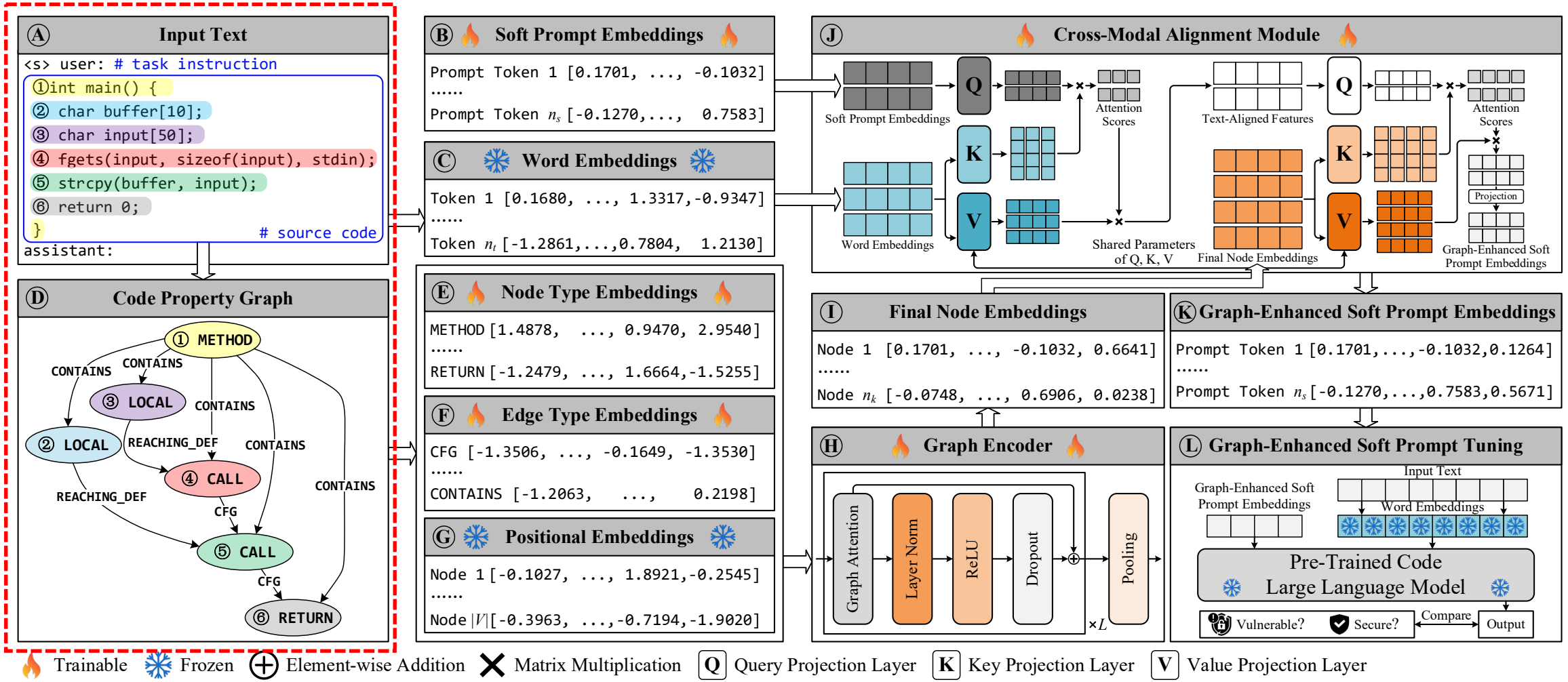
Our Contributions

- Solution to Limitation 1:
  - **Efficient cross-modal alignment module** with a **low computational load** ($|V| + N$) while considering the **graph-text interactions**.
- Solution to Limitation 2:
  - **Type-aware embeddings** that **explicitly** capture the rich semantics in **node and edge types** of a code property graph.

Structure-Aware Soft Prompt Tuning (CGP-Tuning)



Structure-Aware Soft Prompt Tuning (CGP-Tuning)

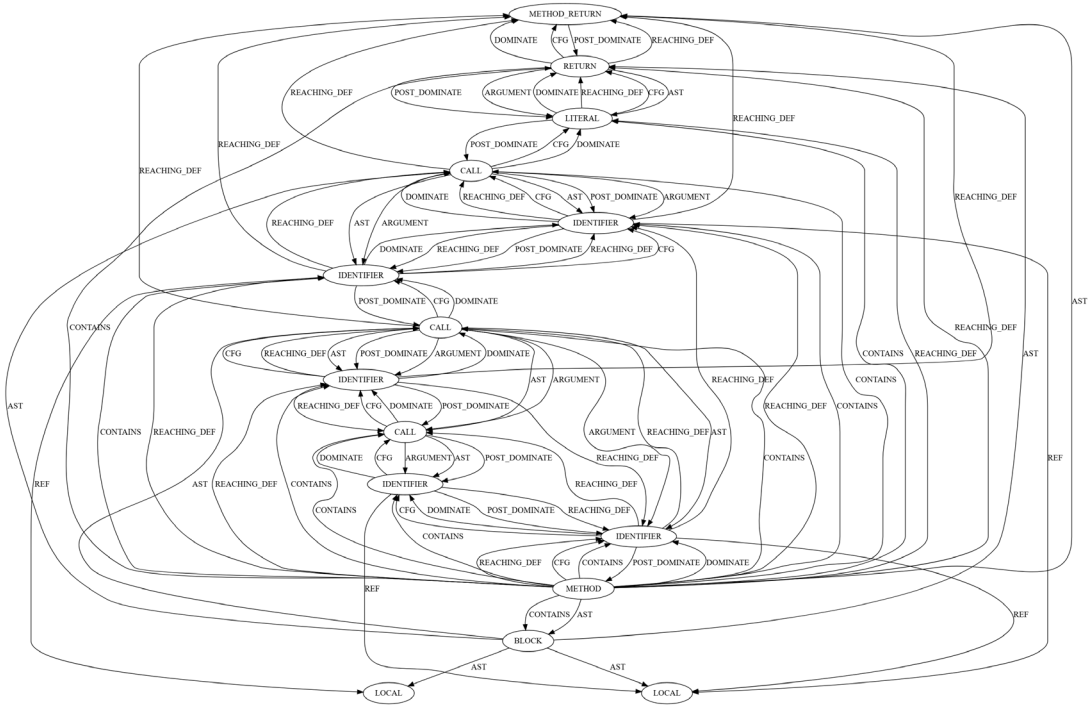


Code Property Graph

- **Code property graph** describes the **syntax structure**, **control flow**, and **program dependencies** of the source code.

Ⓐ Input Text

```
<s> user: # task instruction
①int main() {
② char buffer[10];
③ char input[50];
④ fgets(input, sizeof(input), stdin);
⑤ strcpy(buffer, input);
⑥ return 0;
}                                  # source code
assistant:
```



Too Complicated

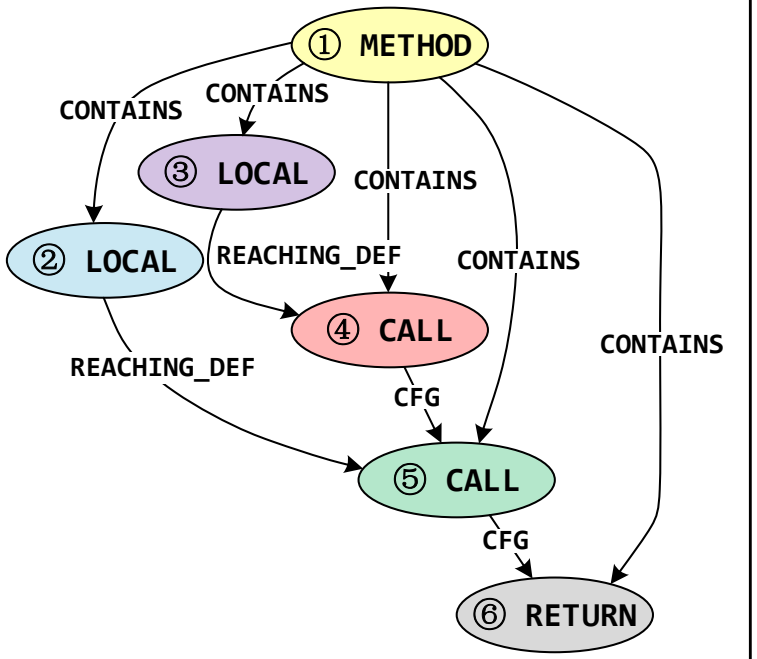
Code Property Graph

- Each code statement is **abstracted** into a node in the graph. The graph provides the **structural information** of the source code.

Ⓐ **Input Text**

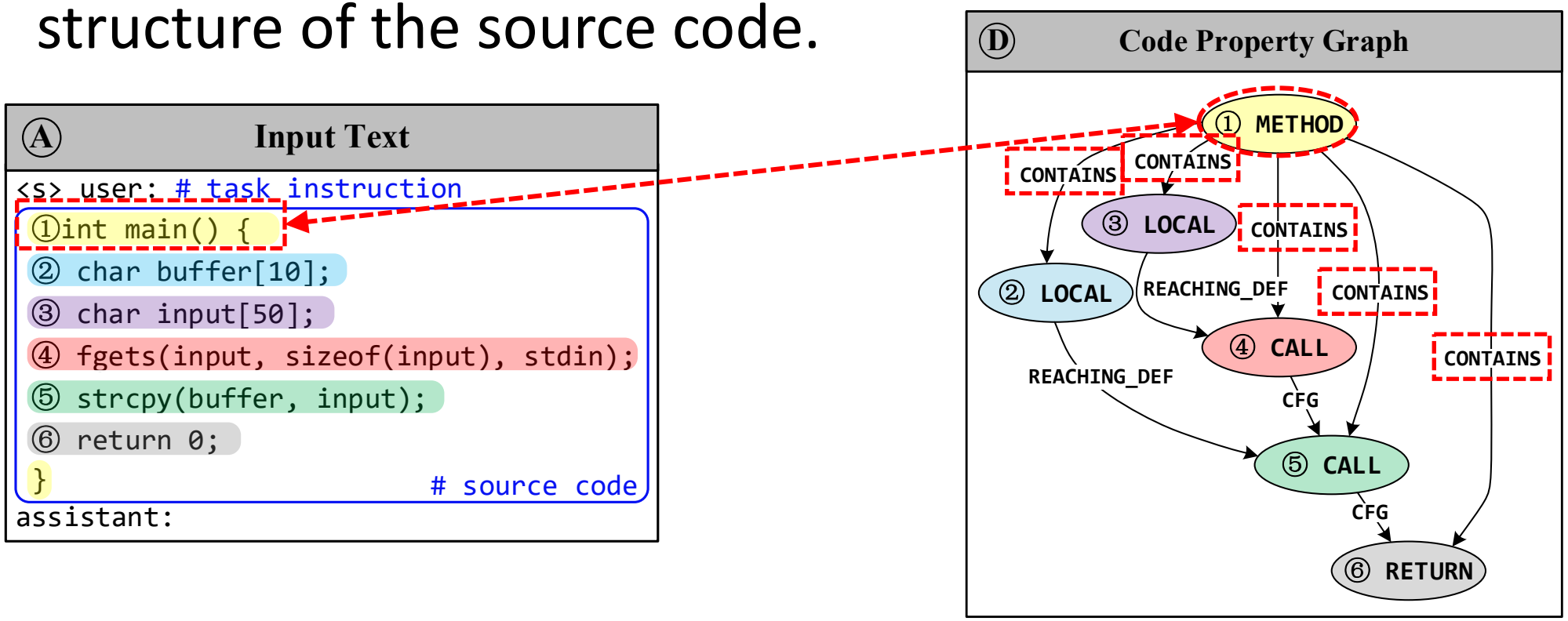
```
<s> user: # task instruction
①int main() {
② char buffer[10];
③ char input[50];
④ fgets(input, sizeof(input), stdin);
⑤ strcpy(buffer, input);
⑥ return 0;
}                                  # source code
assistant:
```

Ⓓ **Code Property Graph**

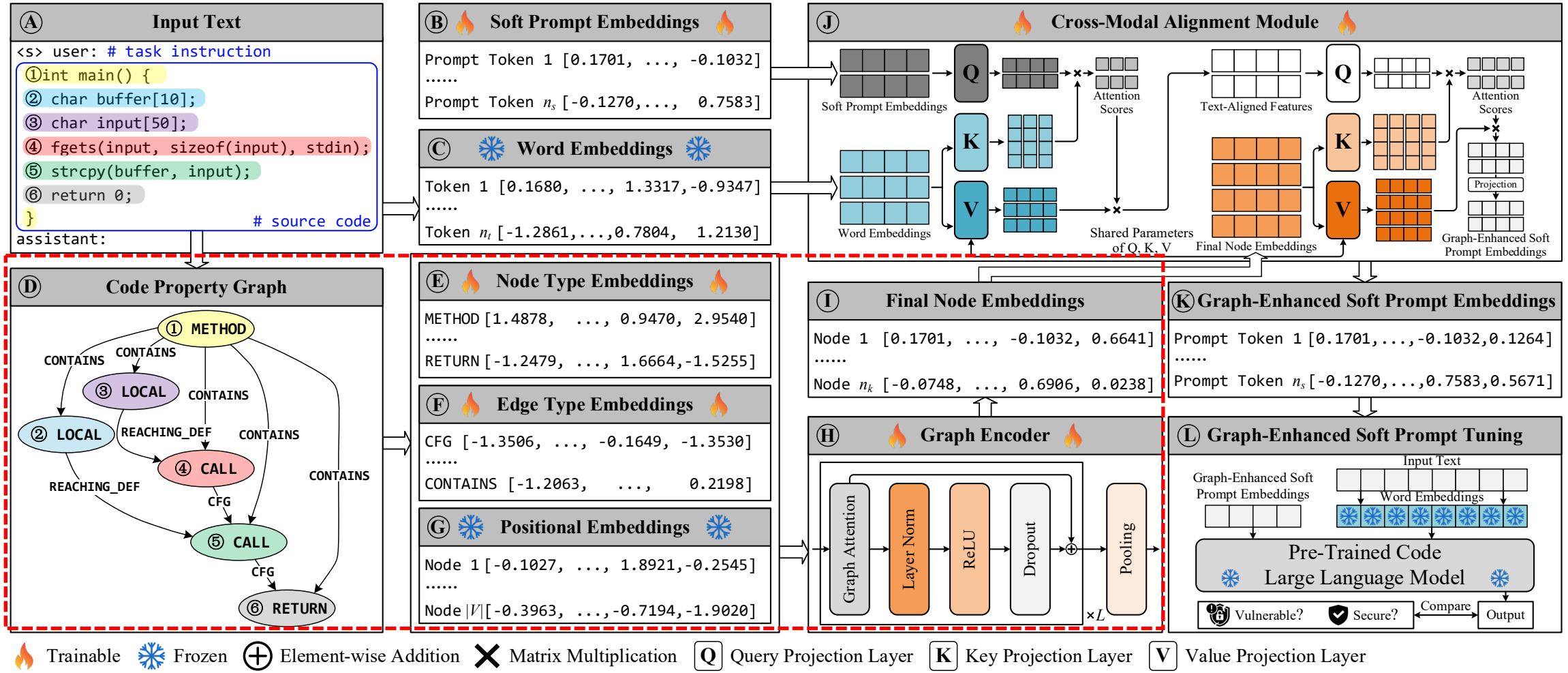


Code Property Graph

- Node 1 corresponds to the declaration of the main function that contains all the code statements. Thus, node 1 have edges point to all other nodes, describing the syntax structure of the source code.

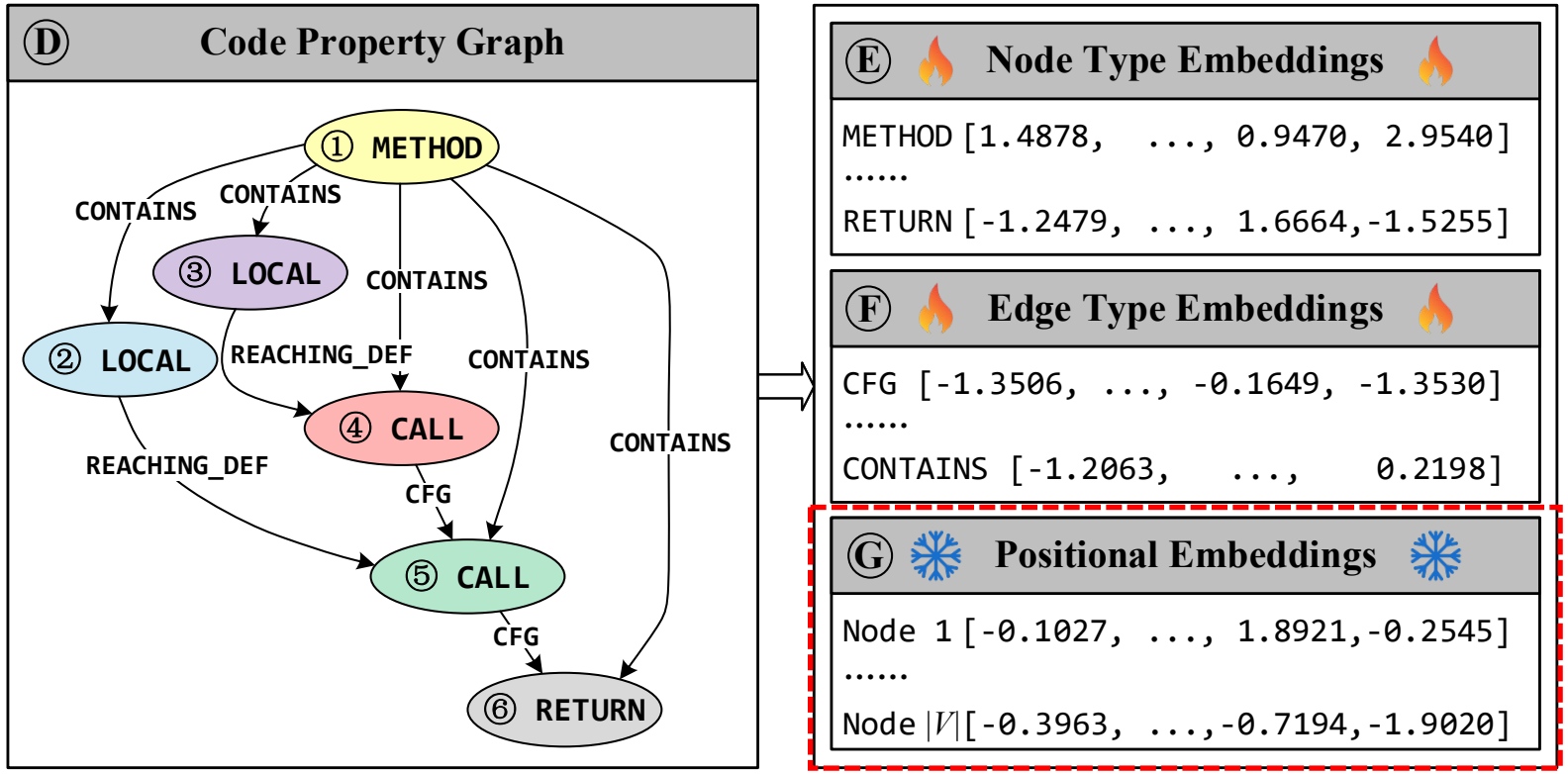


Structure-Aware Soft Prompt Tuning (CGP-Tuning)



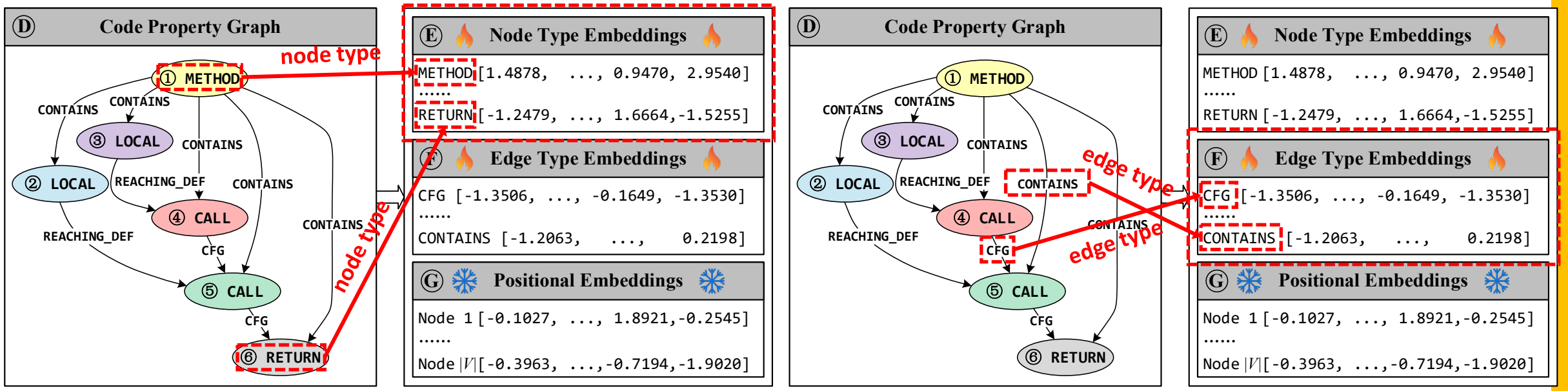
Graph Representation—Node Embeddings

- Each node is represented with sinusoidal positional embedding as initial node embedding, which ensures that the representation of each node is unique.



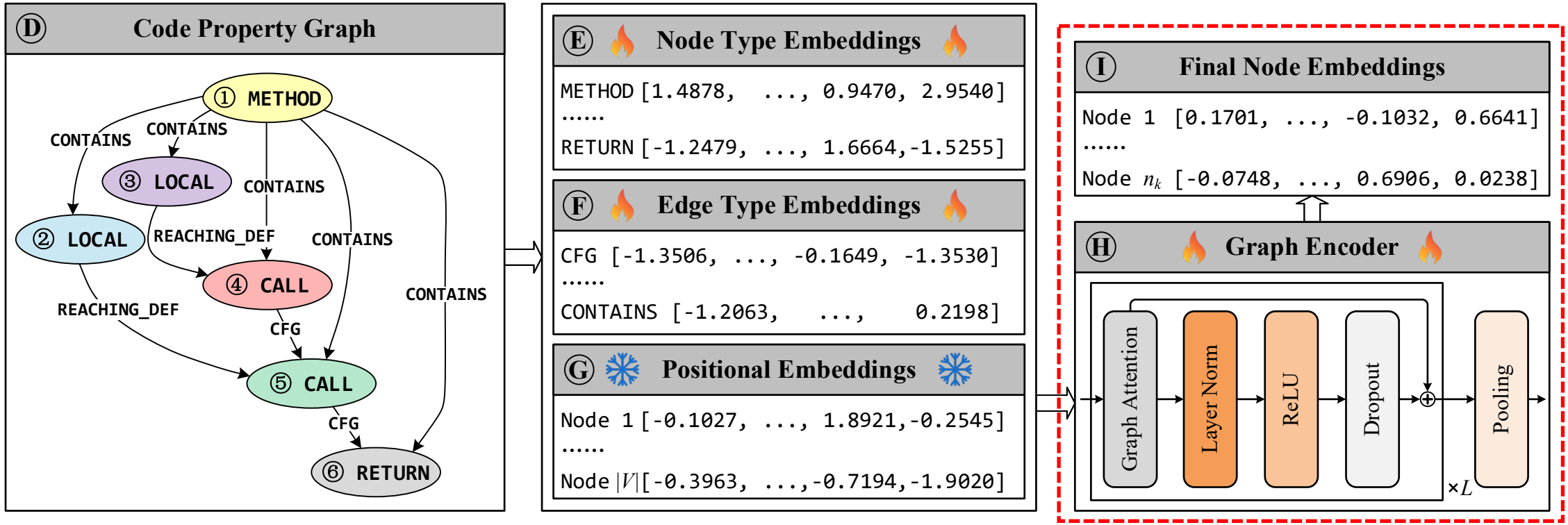
Graph Representation—Type-Aware Embeddings

- **Edge** and **node** type information are represented by **trainable type-aware embeddings**, to **explicitly** represent the edge type and node type information of code property graph.

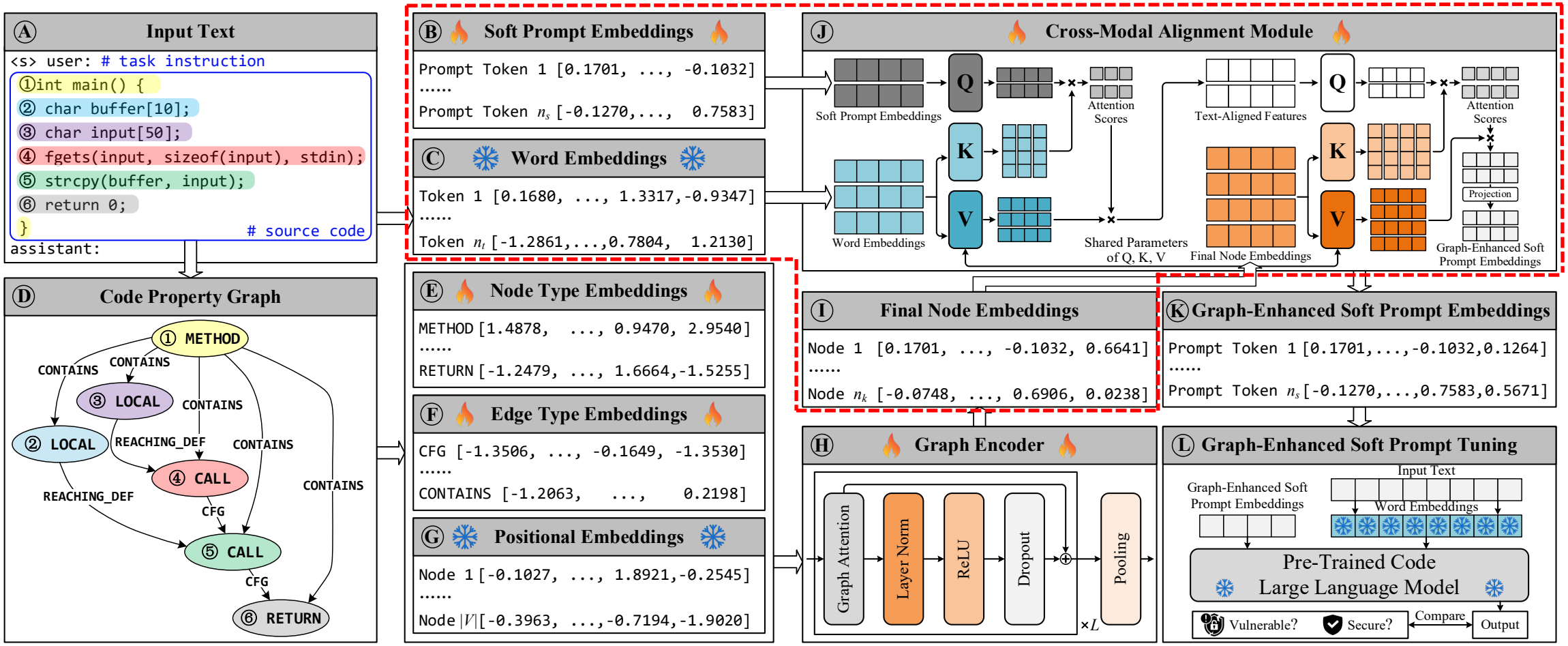


Graph Encoder

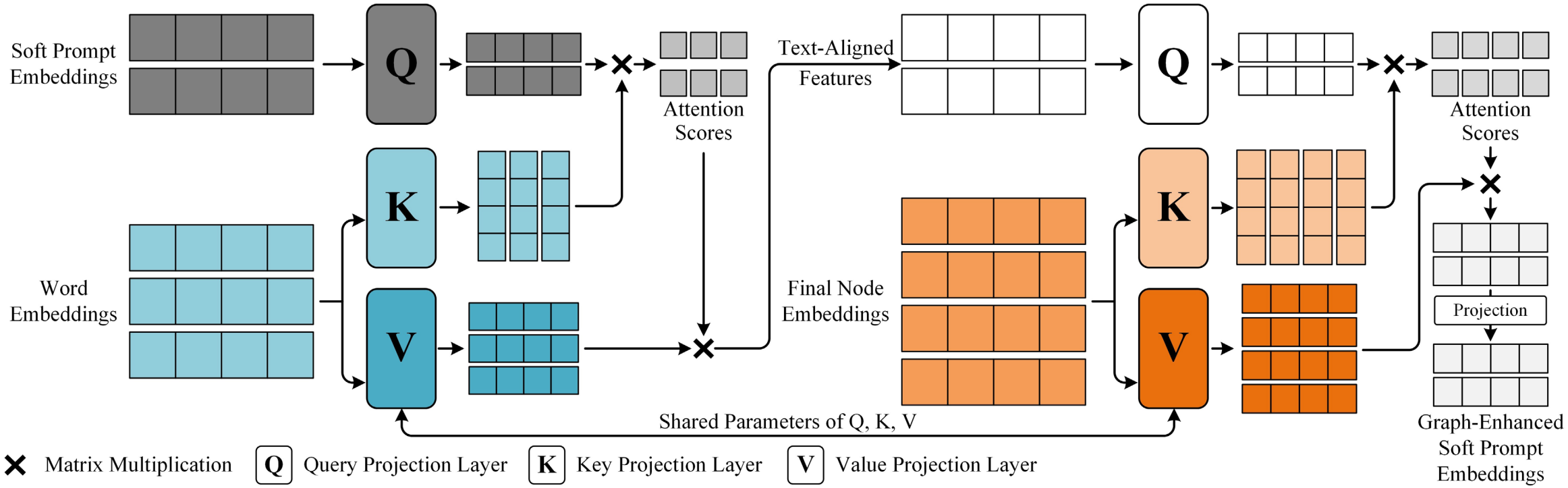
- Graph encoder takes them as input, and produce the **final node embeddings** as the **graph features**



Structure-Aware Soft Prompt Tuning (CGP-Tuning)

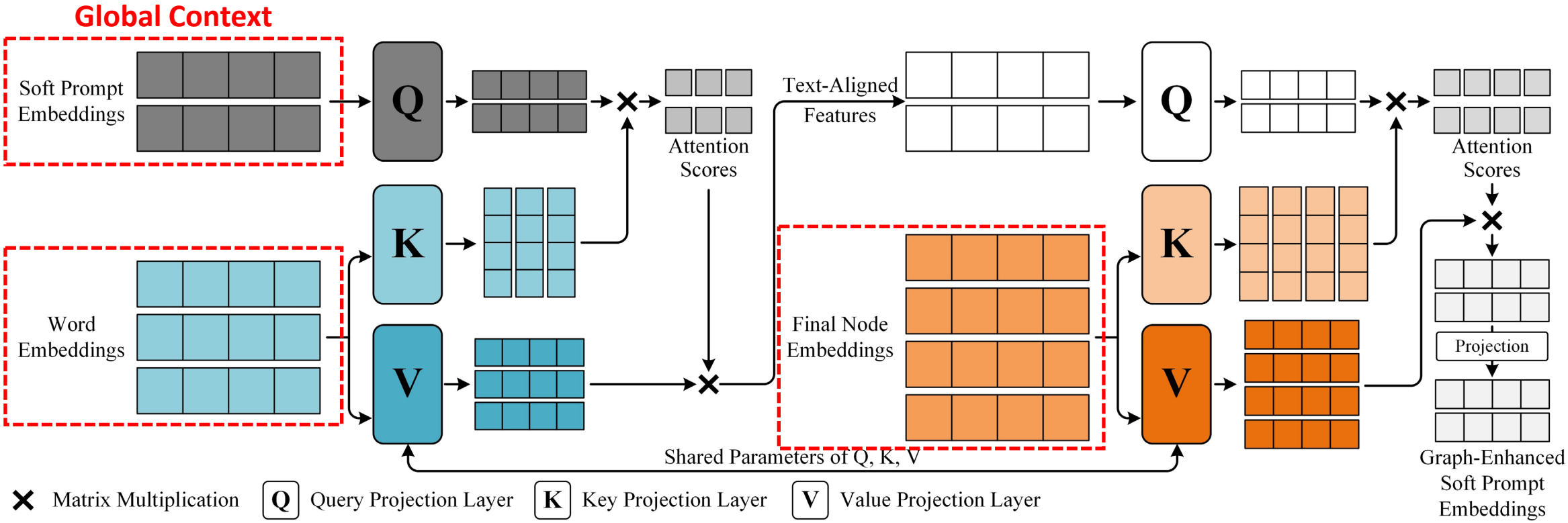


Efficient Cross-Modal Alignment Module



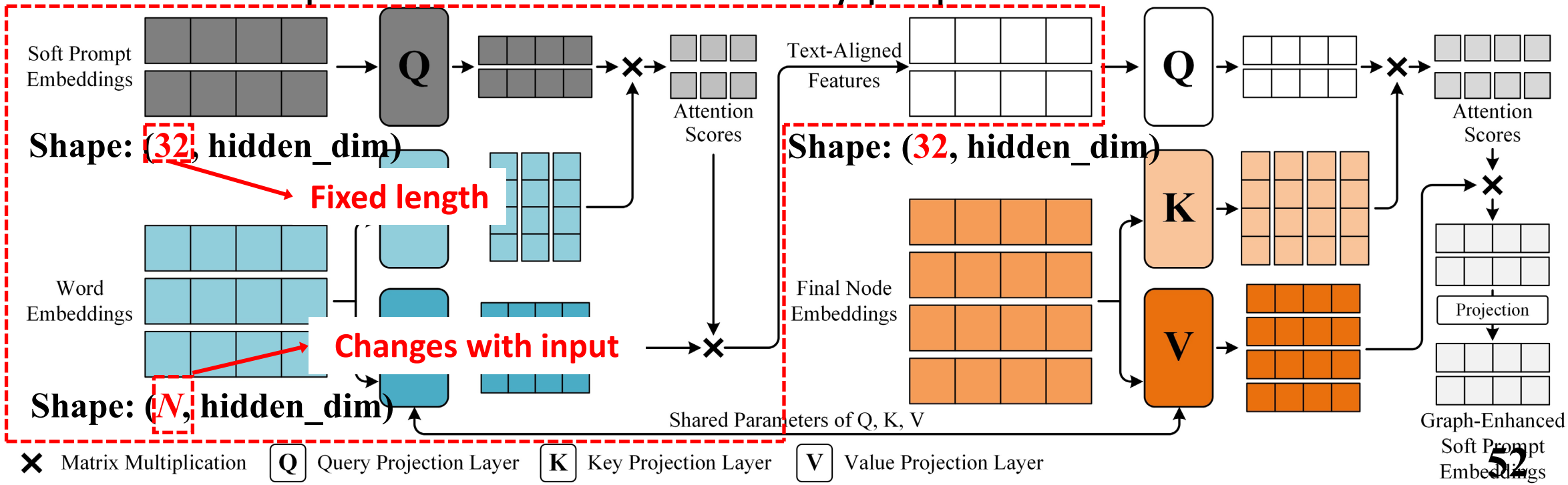
Efficient Cross-Modal Alignment Module

- Use the soft prompt embeddings as a **global context** without directly calculating the relationships between word embeddings and node embeddings.



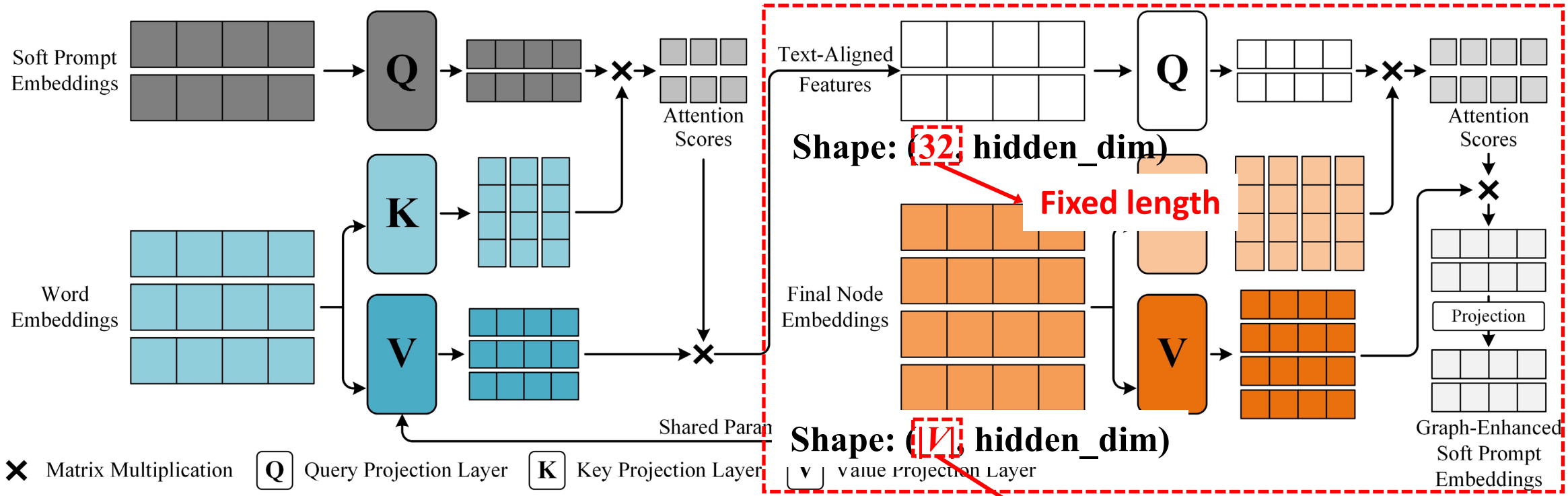
Efficient Cross-Modal Alignment Module

The soft prompt embeddings first perform cross-attention with the word embeddings to obtain **text-aligned features**. Because the length of soft prompt embeddings is **fixed** during **fine-tuning and inference**, so the computational load here is only proportional to N.

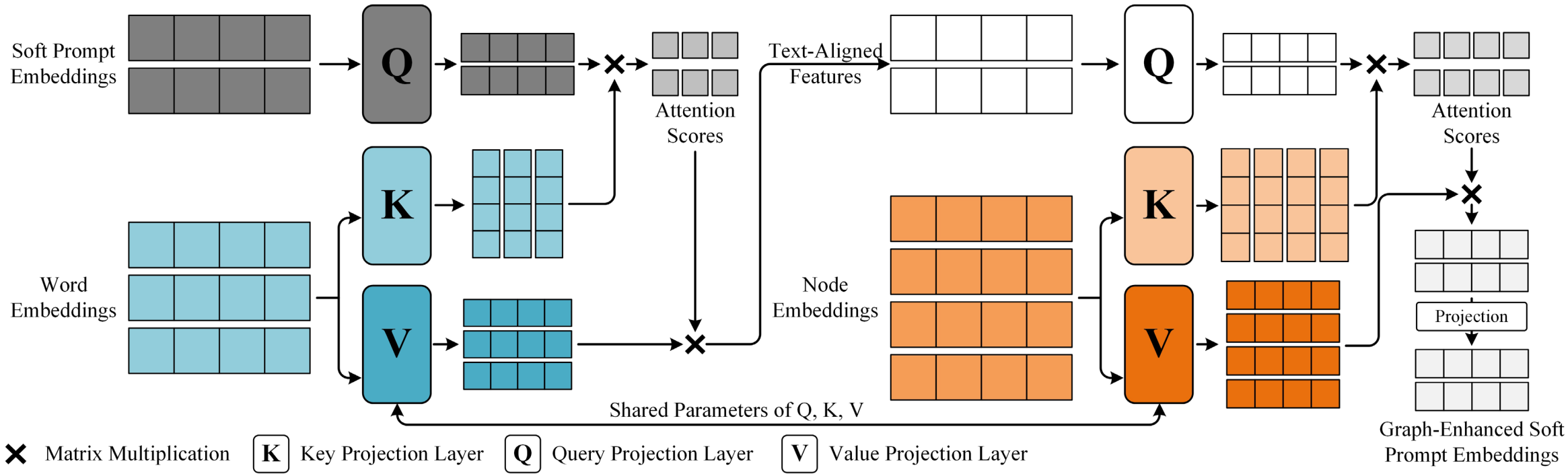


Efficient Cross-Modal Alignment Module

Then, the **text-aligned features** perform cross-attention with the **final node embeddings** to obtain the **graph-enhanced prompt embeddings**. And the computational load is proportional to $|V|$.



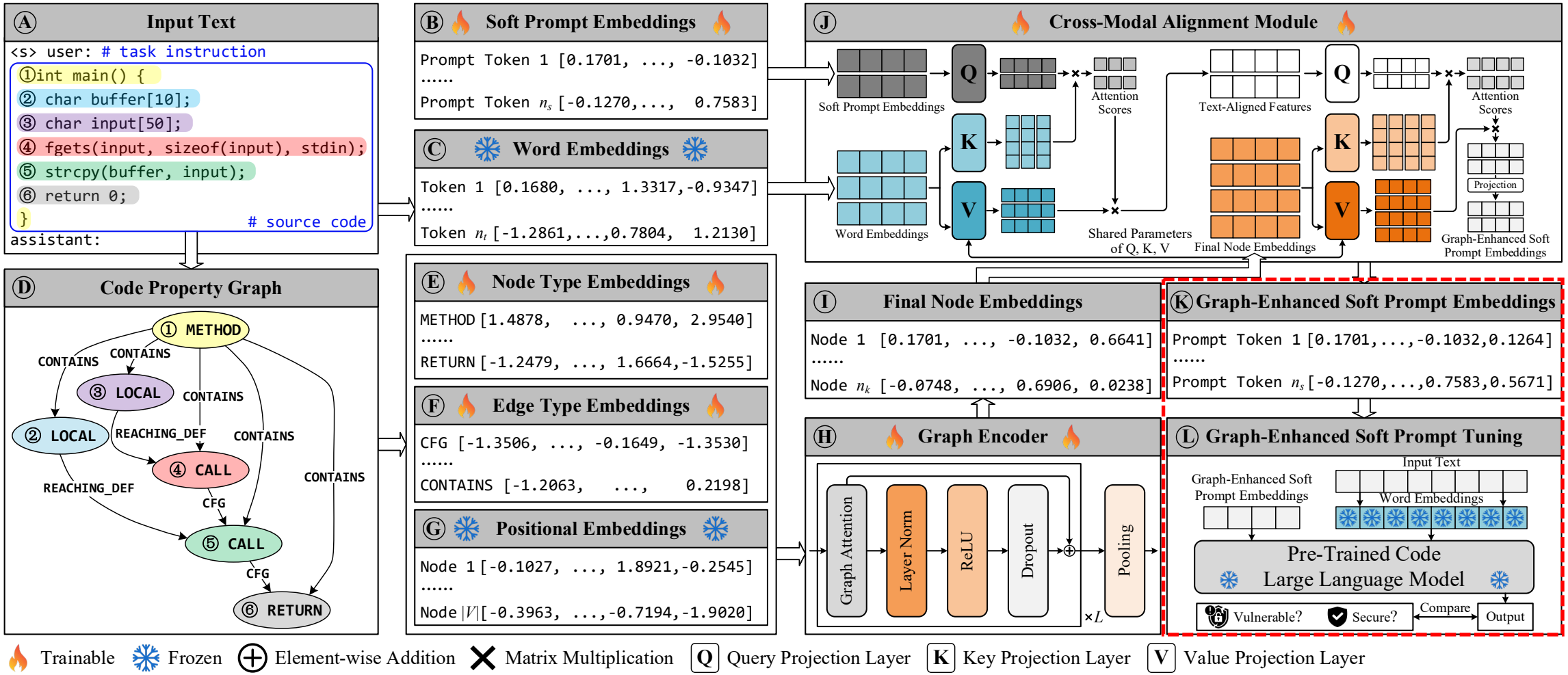
Efficient Cross-Modal Alignment Module



Linear computational load proportional to $|V| + N$

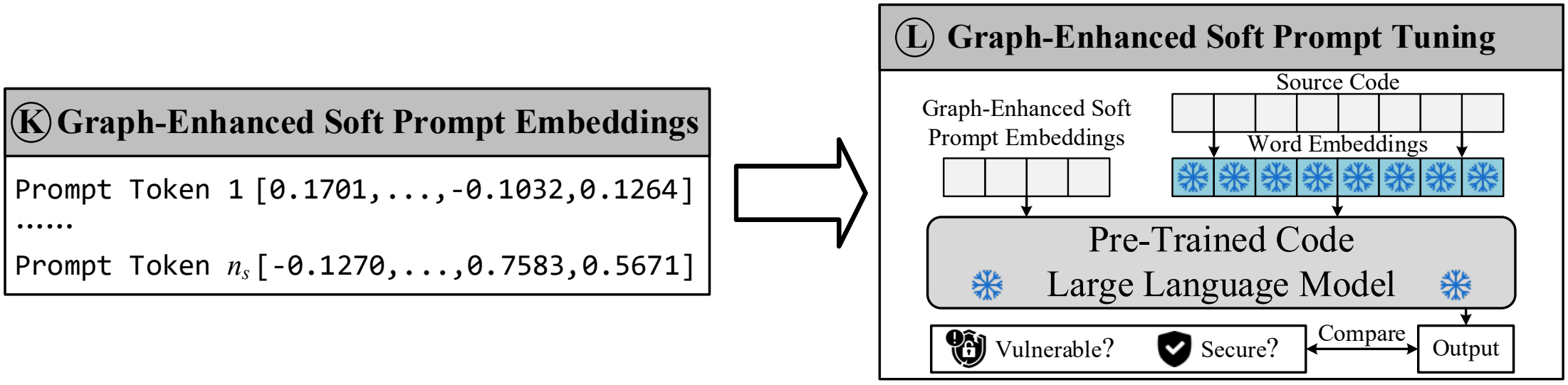
Graph-text interactions ✓

Structure-Aware Soft Prompt Tuning (CGP-Tuning)



Graph-Enhanced Soft Prompt Tuning

- **Graph-enhanced soft prompt embeddings** are used to fine-tune the pre-trained code LLM for **improved vulnerability detection**.



Experiments

Dataset: DiverseVul 33 million samples.

Model: CodeLlama-7B, CodeGemma-7B.

Metrics: Accuracy, Weighted Precision, Weighted Recall, and Weighted F1-score.

Baselines

In-context learning based methods:

- Zero-Shot Prompting — zero-shot ability
- GRACE — in-context learning ability

Tuning based methods:

- Prompt Tuning — standard soft prompt tuning
- G-Retriever — projector-based method
- Graph Neural Prompting (GNP) — cross-attention-based method

Input-Output Example

In-context learning based methods:

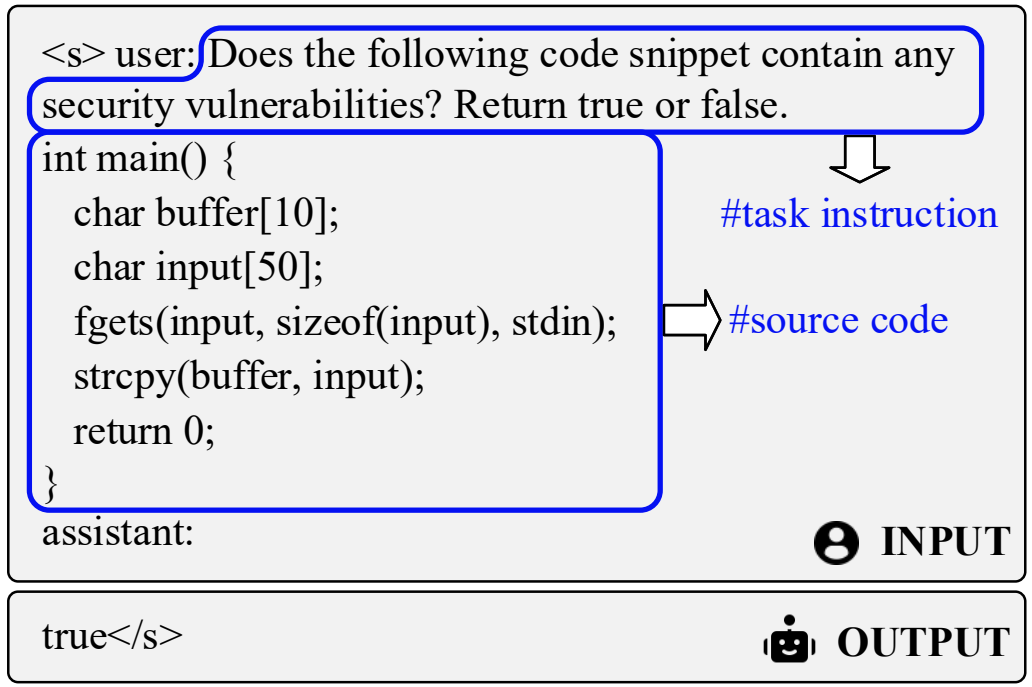
- Zero-Shot Prompting — zero-shot ability
- GRACE — in-context learning ability

Tuning based methods:

- Prompt Tuning — standard soft prompt tuning
- G-Retriever — projector-based method
- Graph Neural Prompting (GNP) — cross-attention-based method

Input-Output Example

The **instruction** in the input tell the model to classify whether the given **code snippet** contain any vulnerabilities and return **true or false**.



Input-Output Example of GRACE

In-context learning based methods:

- Zero-Shot Prompting — zero-shot ability
- GRACE — in-context learning ability

Tuning based methods:

- Prompt Tuning — standard soft prompt tuning
- G-Retriever — projector-based method
- Graph Neural Prompting (GNP) — cross-attention-based method

Lu et al. GRACE: Empowering LLM-based software vulnerability detection with graph structure and in-context learning. Journal of Systems and Software 2024 (Core A).

Input-Output Example of GRACE

Use **natural language** to describe the code property graph and feed the **node and edge information** as extra **context**.

<s> user: #task instruction
#source code
The code property graph of this code is as follows:
Node information: Node ID Node Type
8 METHOD
.....
20 LOCAL #node information

Edge information: Source Target Edge Type
8 10 CONTAINS
.....
10 17 REACHING_DEF #edge information
assistant:

INPUT

true</s>

OUTPUT

Results—Effectiveness of Each Component

Table 1 Ablation study on CodeLlama

	Accuracy	Precision	Recall	F1-Score
CGP-Tuning w/o cross-modal alignment module	0.4752	0.4050	0.4752	0.3563
CGP-Tuning (w/o node type embedding)	0.6512	0.6533	0.6512	0.6500
CGP-Tuning (w/o edge type embedding)	0.6611	0.6613	0.6611	0.6610
CGP-Tuning	**0.6627**	**0.6629**	**0.6627**	**0.6626**

Table 2 Ablation study on CodeGemma

	Accuracy	Precision	Recall	F1-Score
CGP-Tuning w/o cross-modal alignment module	0.4752	0.4050	0.4752	0.3563
CGP-Tuning (w/o node type embedding)	0.6512	0.6533	0.6512	0.6500
CGP-Tuning (w/o edge type embedding)	0.6611	0.6613	0.6611	0.6610
CGP-Tuning	**0.6627**	**0.6629**	**0.6627**	**0.6626**

Results—Comparison with Other Methods

Table 3 Performance comparison on CodeLlama

	Accuracy	Precision	Recall	F1-Score
Zero-Shot Prompting	0.5060	0.6073	0.5060	0.3535
GRACE	0.5958	0.6427	0.5958	0.5596
Prompt Tuning	0.5901	0.6539	0.5901	0.5426
G-Retriever	0.6266	0.6526	0.6266	0.6099
GNP	0.6356	0.6359	0.6356	0.6355
CGP-Tuning	**0.6582**	**0.6590**	**0.6582**	**0.6577**

Table 4 Performance comparison on CodeGemma

	Accuracy	Precision	Recall	F1-Score
Zero-Shot Prompting	0.4681	0.4605	0.4681	0.4410
GRACE	0.4020	0.4003	0.4020	0.3995
Prompt Tuning	0.5113	0.5121	0.5113	0.5026
G-Retriever	0.5692	0.6077	0.5692	0.5269
GNP	0.5755	0.5769	0.5755	0.5736
CGP-Tuning	**0.6219**	**0.6362**	**0.6219**	**0.6118**

Future Work

Limitations and Future Work

Limitation: for code written in **different programming languages** but with **similar semantics**, the graph representation may vary significantly, which may impact performance.

Limitations and Future Work

Limitation: for code written in **different programming languages** but with **similar semantics**, the graph representation may vary significantly, which may impact performance.

Solution: a unified graph representation.

Limitations and Future Work

Limitation: the **input size** that a code LLM can handle is quite **limited (e.g., 64K)**, due to the **O(N^2) complexity** of self-attention and hardware constrains, making it challenging to scale up to large, real-world projects.

Limitations and Future Work

Limitation: the **input size** that a code LLM can handle is quite **limited (e.g., 64K)**, due to the **O(N^2) complexity** of self-attention and hardware constrains, making it challenging to scale up to large, real-world projects.

Solution: develop better **token compression** and **pruning** methods to reduce input tokens while retaining the essential **semantic** and **structural** information of the code.

Thank you

Q&A