NAVER: A Neuro-Symbolic Compositional Automaton for Visual Grounding with Explicit Logic Reasoning

Supplementary Material

This supplementary material provides additional details about the proposed method NAVER. In particular, it provides details about the additional quantitative comparison result with both accuracy and IoU in section 6, the complexity analysis in section 7, the ablation studies for VLM in section 8, the analysis of self-correction mechanisms in section 9, the performance analysis for query length in section 10, the ablation studies for captioner in section 11, and the prompts used for the LLMs and VLMs in section 12.

6. Additional Quantitative Comparison

In this section, we report both the accuracy and Intersection over Union (IoU) performance for the referring expression detection task. Table 9 presents a quantitative comparison on the RefCOCO, RefCOCO+, RefCOCOg [21], and RefAdv [1] datasets. Additionally, we include results for the compositional methods using GroundingDINO as the VFM, marked with ‡. We observed that for each configuration, our proposed method NAVER outperforms the other compositional methods using the same foundation models and also outperform the grounding methods themselves. These results further validate that our approach consistently achieves superior IoU performance over existing baselines.

7. Complexity Analysis

Efficiency results for NAVER and compositional baselines are provided in Table 10, which includes average runtime, token usage, and cost per sample. For a fair comparison, we use the same foundation models and LLM for the experiments. From the results, we observe NAVER is more efficient than HYDRA [22] with higher performance and lower runtime failure rate. Although ViperGPT [45] has lower time complexity, this advantage is offset by reduced performance due to its lack of validation. These results demonstrate that NAVER's flexible automaton design reduces unnecessary computations by dynamically adapting the pipeline.

8. Ablation Studies for VLMs

We evaluate the impact of four different Vision-Language Models (VLMs) [10, 26, 30, 54] on the performance of NAVER. The results are shown in Table 11. All tested models achieve comparable results on both RefCOCO and RefCOCO+. Among the four VLMs, InternVL2-8B achieves slightly better performance with accuracy scores of 96.2 on RefCOCO and 92.8 on RefCOCO+. However, the differences between the VLMs are marginal, indicating that the architecture of NAVER is insensitive to the choice of VLM. This shows NAVER's ability to decompose complex tasks into simpler subtasks, reducing the reliance on the capabilities of the VLMs. Given the similar performance across all tested VLMs, we select InternVL2-8B for its availability and accessibility, ensuring simpler implementation without compromising performance. This demonstrates the flexibility of NAVER in adopting different VLMs while delivering state-of-the-art performance.

9. Self-Correction Analysis

We analyze the performance of the self-correction mechanism in NAVER, focusing on its ability to address errors during inference. In this mechanism, each time the system transitions into the self-correction state (indicated by a red arrow in Figure 2), it is counted as one retry. Our experiments, conducted on the RefCOCO test A dataset, show that approximately 10% of the samples require self-correction. To better understand the behavior of the mechanism, we calculate the proportion of samples resolved after each number of retries within the subset of data requiring self-correction. These results are visualized as a bar chart in Figure 3. The analysis shows that a single retry resolves 68% of the errors, and 96% of the errors are resolved within six retries. To prevent infinite looping, we limit the maximum number of retries to six, as this threshold effectively addresses the majority of the errors.

10. Query Length Analysis

We analyze the impact of text query length on performance, comparing NAVER with baselines across different query lengths. The results, shown in Figure 4, indicate that NAVER consistently reaches SoTA performance compared to all baselines regardless of query length. While most baseline methods experience a performance decline as query length increases, the extent of decrease varies. Notably, older models like GLIP [27] show a significant performance drop for longer queries, suggesting difficulties in handling complex text inputs. In contrast, NAVER maintains stable performance, showing its robustness in processing longer and more complex queries.

	Method	RefCOCO		RefCOCO+		RefCOCOg		Ref-Adv	
		Acc.	IoU	Acc.	IoU	Acc.	IoU	Acc.	IoU
E2E	GLIP-L [27]	55.0	54.1	51.1	51.3	54.6	54.8	55.7	55.2
	KOSMOS-2 [39]	57.4	-	50.7	-	61.7	-	-	-
	YOLO-World-X [11]	12.1	12.7	12.1	12.7	32.9	33.8	32.2	34.2
	YOLO-World-V2-X [11]	19.8	20.0	16.8	17.2	36.5	37.3	33.1	34.8
	GroundingDINO-T [32]	61.6	60.2	59.7	58.9	60.6	59.7	60.5	59.8
	GroundingDINO-B [32]	90.8	85.2	84.6	77.5	80.3	69.4	78.0	73.1
	SimVG [13]	94.9	86.9	91.0	83.9	88.9	81.3	74.4	70.7
	Florence2-B [53]	94.5	89.5	91.2	86.5	88.3	85.0	72.2	71.9
	Florence2-L [53]	95.1	90.6	92.5	88.2	90.9	87.6	71.8	71.8
Compositional	Code-bison [44]	44.4	-	38.2	-	-	-	-	-
	ViperGPT† [45]	62.6	59.4	62.3	58.7	67.2	63.6	60.7	58.6
	HYDRA† [22]	60.9	58.2	56.5	54.9	62.9	60.8	54.4	53.5
	NAVER†	**70.1**	**67.9**	**64.1**	**63.8**	**69.5**	**59.2**	**65.1**	**63.4**
	ViperGPT‡	67.1	64.0	73.2	68.8	65.6	63.5	60.1	59.5
	HYDRA‡	63.5	61.3	62.9	60.9	59.8	58.4	53.2	53.5
	NAVER‡	**91.7**	**83.3**	**82.4**	**78.4**	**75.9**	**72.5**	**87.3**	**74.2**
	ViperGPT*	68.6	66.5	73.8	70.4	68.7	66.8	58.2	58.3
	HYDRA*	65.7	64.6	66.2	65.3	59.9	60.5	48.3	51.8
	NAVER*	**96.2**	**91.7**	**92.8**	**88.4**	**91.6**	**88.1**	**75.4**	**75.3**

Table 9. **Accuracy and IoU performance on the referring expression detection task.** Results are shown on the RefCOCO, RefCOCO+, RefCOCOg [21], and Ref-Adv [1] datasets. In compositional methods, they are grouped with the same VLMs for fair comparison. The methods with same symbols (†, ‡, *) use the same VFMs. The VFMs used are: (†) uses GLIP-L and BLIP; (‡) uses GroundingDINO-B and InternVL2; (*) uses Florence2-L and InternVL2, respectively. All groups use GPT-4o Mini.

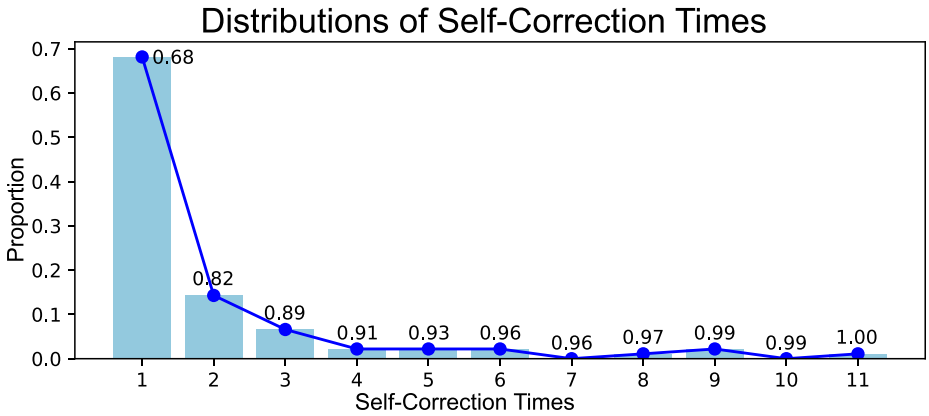

Figure 3. **Distribution of self-correction times in NAVER.** The x-axis represents the number of retries for self-correction, while the y-axis shows the proportion of samples requiring self-correction within each group of self-correction times. The annotated values indicate the cumulative proportion of samples resolved by that number of retries.

Method	Time (Seconds)	# LLM Tokens Input	# LLM Tokens Output	LLM Cost (USD)
ViperGPT [45]	**2.02**	4169	**29**	0.00064
HYDRA [22]	14.93	19701	618	0.00332
NAVER (Ours)	5.96	**2083**	39	**0.00034**

Table 10. **Complexity comparison between compositional methods.** All results are evaluated on the RefCOCOg test set with the GPT-4o Mini for fair comparison. All values are the average per data sample. All methods use the same foundation models.

VLM	RefCOCO	RefCOCO+
BLIP2-XXL [26]	95.4	91.7
xGen-MM [54]	94.5	87.9
LLaVA1.5-7B [30]	95.5	91.9
InternVL2-8B [10]	**96.2**	**92.8**

Table 11. **Comparison of VLM selection for NAVER.** All results (accuracy) are evaluated on the RefCOCO and RefCOCO+ testA datasets [21].

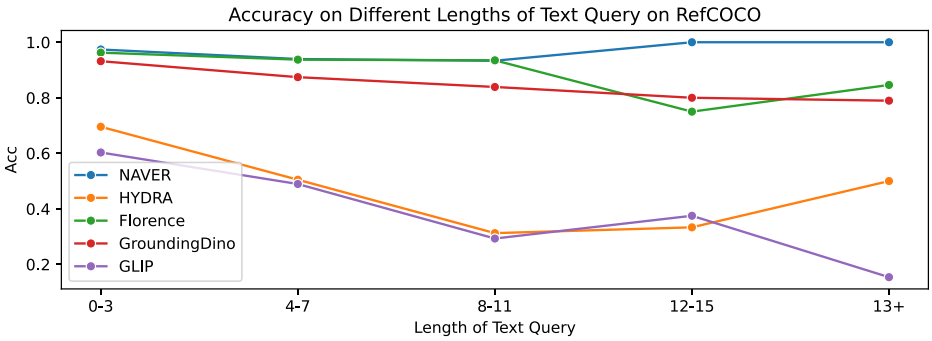

Figure 4. **Accuracy performance of different text query length in NAVER.** The x-axis represents the length of text query, while the y-axis shows the accuracy of NAVER and baselines within each group of query length.

Captioner	ECE	RefCOCO	RefCOCO+	RefCOCOg	RefAdv
✗	VLM	91.6	87.5	80.2	65.6
VLM	LLM	**96.2**	**92.8**	**91.6**	**75.4**

Table 12. **Ablations for the captioner and type of entity category extractor (ECE).** All results are accuracy for referring expression detection task.

11. Ablations for Captioner

In the perception state, NAVER first converts the image into a rich caption and then lets an LLM-based entity-category extractor (ECE) decide which object classes are relevant to the query. An intuitive alternative is to skip captioning and ask the VLM to predict categories directly. Table 12 shows that this seemingly simpler choice causes a consistent accuracy drop of 4.6% on RefCOCO, 5.3% on RefCOCO+, 11.4% on RefCOCOg, and 9.8% on Ref-Adv. The larger gap on the more complex datasets (RefCOCOg, Ref-Adv) suggests that a textual scene summary helps the downstream logic generator reason about complex descriptions. The clear improvement brought by the Captioner justifies keeping it in the final system.

12. LLM and VLM Prompts

NAVER employs LLMs and VLMs in five different roles as described in section 3: as a caption generator VLM in the *Perception* state, an entity extractor LLM in the *Perception* state, a logic query generator LLM in the *Logic Generation* state, a relation recognizer VLM in the *Logic Generation* state, and an answerer in the *Answering* state. The prompts utilized for each role are carefully crafted to generate precise outputs aligned with the requirements of each state, ensuring accurate visual grounding and robust reasoning. Prompts 12.1 to 12.5 show the detailed prompt templates for each role.

- Prompt 12.1: This prompt instructs the VLM to generate descriptive captions I_c for the input image I. The generated captions represent the visual content in textual form, which serves as input for the entity extractor LLM in subsequent processing.
- Prompt 12.2: This prompt is utilized in the *Perception* state to guide the LLM in identifying entity categories C relevant to the query Q and the caption I_c. The output is a list of entity categories, which forms the basis for visual reasoning in the subsequent states.
- Prompt 12.3: This prompt guides the logic query generator LLM to convert the textual query Q into a ProbLog logic query in the *Logic Generation* state. The output logic query is then utilized in the ProbLog interpreter to perform probabilistic logic reasoning.
- Prompt 12.4: This prompt guides the relation recognizer VLM to identify relations R between entities E in the *Logic Generation* state. These recognized relations are used to construct logic expressions, as the foundation for probabilistic logic reasoning.
- Prompt 12.5: This prompt is used in the *Answering* state to guide the answerer VLM in validating whether the identified target Y_{L1} satisfies the conditions of the query Q. It requests a binary response ("Yes" or "No") to confirm whether the top-ranked candidate from the *Logic Reasoning* state fulfills all query requirements, ensuring only valid results are returned as final outputs.

Prompt 12.1: Captioner VLM

⟨image⟩Please describe the image in detail.

Prompt 12.2: Entity Extractor LLM

You're an AI assistant designed to find detailed information from image.

You need to find important objects based on the given query which is the object you need to find. The query normally is a set of words which includes a object name and the attributes of the object.

Here are some examples:
Query: ⟨example query 1⟩
Answer: ⟨example answer 1⟩

Query: ⟨example query 2⟩
Answer: ⟨example answer 2⟩
...
Your output must be a JSON object contains the flatten list of string. For example: "output": ["apple", "orange", "chair", "umbrella"]

Caption: ⟨caption⟩
Query: ⟨query⟩
Answer:

Prompt 12.3: Logic Query Generator LLM

You're an AI assistant designed to generate the ProbLog code (a logic programming language similar to Prolog).

You need to generate a new rule "target" that will be used to query the target objects in the image based on given text prompt.

The names of entity categories are ⟨entity categories⟩.

The output is the code. For example:
```problog
target(ID) :- entity(ID, "<some category>", _, _, _, _), relation(ID, _, _), attribute(ID, _).
```

More examples:
find the target "⟨example query 1⟩"
```problog
⟨example ProbLog code 1⟩
```

find the target "⟨example query 2⟩"
```problog
⟨example ProbLog code 2⟩
```
...

Complete the following ProbLog code:
```problog
⟨ProbLog code of context⟩
```

Your output should be the ProbLog code.
find the target "⟨query⟩"
Your answer:

Prompt 12.4: Relation Recognizer VLM

⟨image⟩You're an AI assistant designed to find the relations of objects in the given image.

The interested objects are highlighted by bounding boxes (X1, Y1, X2, Y2). They are:

A: the ⟨category of A⟩labeled by red bounding box ⟨bbox of A⟩.
B: the ⟨category of B⟩labeled by red bounding box ⟨bbox of B⟩.

Only consider the camera view. Note you are focusing to analyze the relation A to B, do not consider the relation B to A. Please answer "Yes" or "No" for the following question.

Is A ⟨relation⟩ B?

Your answer is:

Prompt 12.5: Answerer VLM

⟨image⟩You're an image analyst designed to check if the highlighted object in the image meets the query description.

The query is: "⟨query⟩"

Please check the highlighted object "A" in the image and answer the question: Does the highlighted object meet the query description? Your answer should be "Yes" or "No".

Your answer: