

GeoX: Mastering Geospatial Reasoning Through Self-Play and Verifiable Rewards

Kyeongjin Ahn[1,3] · Seungeon Lee[1 *] · Krishna P. Gummadi[1] · Meeyoung Cha[2,3]

[1]Max Planck Institute for Software Systems · [2]Max Planck Institute for Security and Privacy · [3]KAIST



MAX PLANCK INSTITUTE
FOR SOFTWARE SYSTEMS

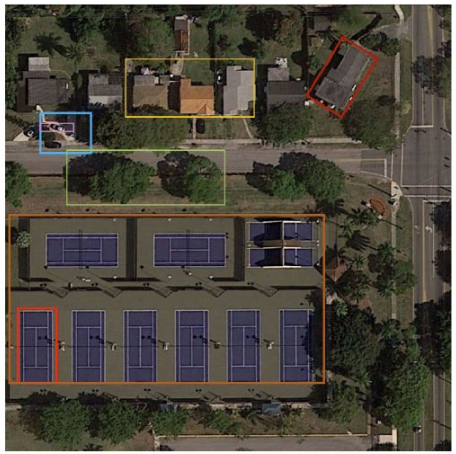
MAX PLANCK INSTITUTE
FOR SECURITY AND PRIVACY



KAIST

Challenges in VLMs for Remote Sensing

Q. " *Is the yellow-box buildings closer to the tennis court than the red-box building?* "



Step 1. Find *yellow-box buildings* + Find *tennis court*
Step 2. Find *red-box building* + Find *tennis court*
Step 3. Compare them

A. " yellow-box buildings "

The practical question require not just ***what*** is present, but ***how*** objects interact.

Challenges in VLMs for Remote Sensing

Challenge 1. RS-VLMs stay bounded to simple recognition, not complex geospatial reasoning.
Challenge 2. Such complex data cannot be labeled at scale, as the question space explodes.

Conventional QA: Simple recognition	Practical QA: Complex geospatial reasoning
Human designed questions (e.g., template, prompt)	Beyond human design
Q. " *Are there <**buildings**>?* " Q. " *How many <**vehicles**> are?* " A. Externally labeled (e.g., Human, Frontier VLMs)	Q. " *Is the largest **land-use area** **greater than** the building **area**?* " Q. " *Which **quadrant** has the **nearest** **parking lot** to **buildings**?* " A. Not labeled
Human-curated QA	Human curation is impossible

Problem Requirements

Challenge 1. RS-VLMs stay bounded to simple recognition, not complex geospatial reasoning.
→ Conventional benchmarks teach only simple questions.

Requirement 1. Cover complex reasoning, not just recognition.

Challenge 2. Such complex data cannot be labeled at scale, as the question space explodes.
→ No one can label every possible question.

Requirement 2. Generate complex problems automatically without human curation.

Use a vision-language model to learn geospatial reasoning
by generating and verifying its own complex problems, with no human-curated QA.

Problem Requirements

Challenge 1. RS-VLMs stay bounded to simple recognition, not complex geospatial reasoning.
→ Conventional benchmarks teach only simple questions.

Requirement 1. Cover complex reasoning, not just recognition.

Challenge 2. Such complex data cannot be labeled at scale, as the question space explodes.
→ No one can label every possible question.

Requirement 2. Generate complex problems automatically without human curation.

Use a vision-language model to learn geospatial reasoning
by generating and verifying its own complex problems, with no human-curated QA.

Problem Requirements

Challenge 1. RS-VLMs stay bounded to simple recognition, not complex geospatial reasoning.
→ Conventional benchmarks teach only simple questions.

Requirement 1. Cover complex reasoning, not just recognition.

Challenge 2. Such complex data cannot be labeled at scale, as the question space explodes.
→ No one can label every possible question.

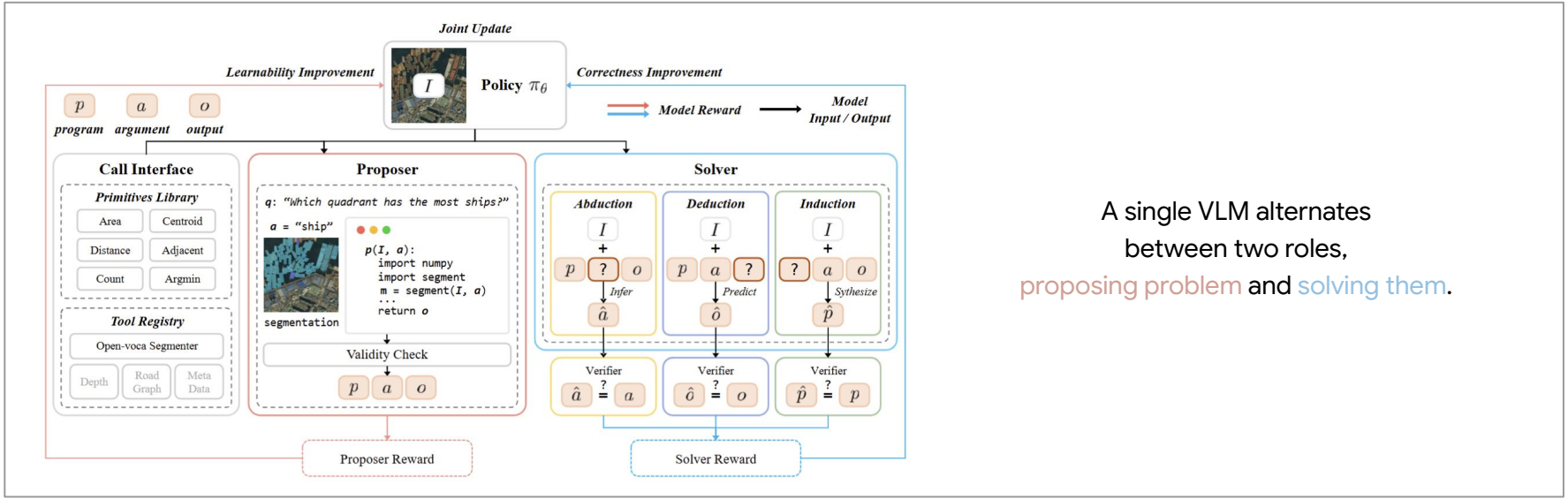
Requirement 2. Generate complex problems automatically without human curation.

Use a vision-language model to learn geospatial reasoning
by generating and verifying its own complex problems, with no human-curated QA.

Our Approach. GeoX (The Self-play Framework for Geospatial Understanding)

Proposer. Construct an executable geospatial problem

Solver. Find the solution to the constructed problem



A single VLM alternates between two roles, proposing problem and solving them.

METHOD

Our Approach. GeoX (The Self-play Framework for Geospatial Understanding)

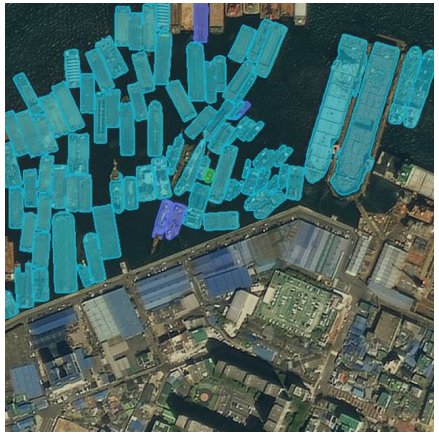
Proposer. Construct an executable geospatial problem

Solver. Find the solution to the constructed problem



Q. Which quadrant has the most ships?

Make it readable!



Raw Image
Rich spatial information is in the image, but not yet usable

Computable Representation
Now usable (e.g., area, centroid, distance, adjacency)

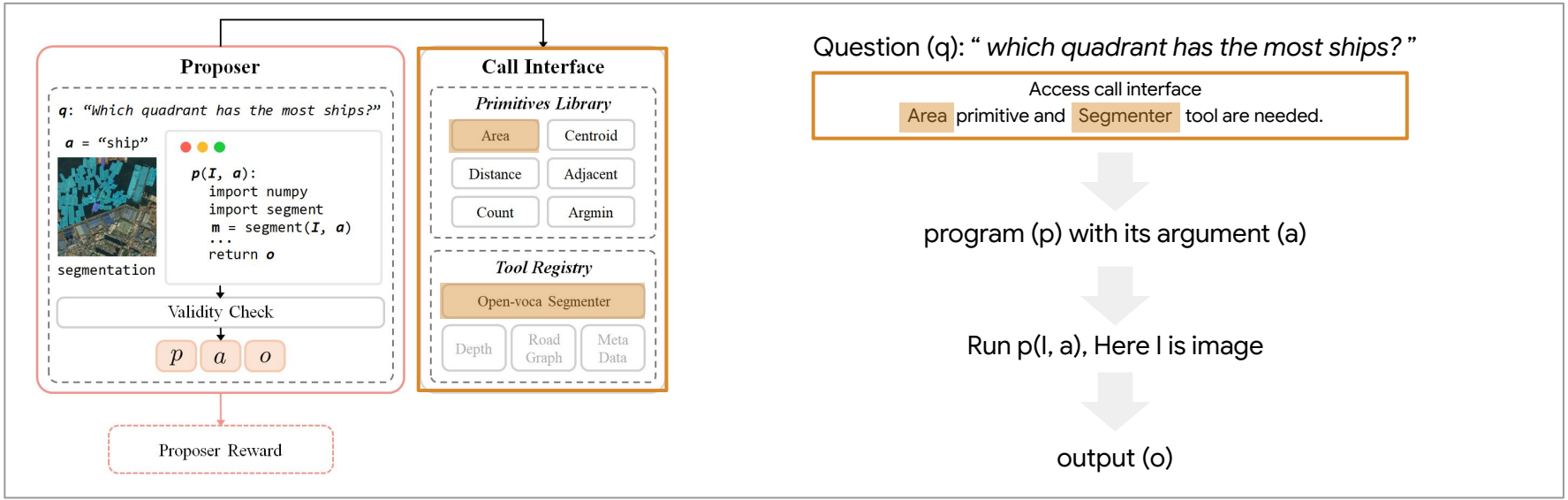
METHOD

Our Approach. GeoX (The Self-play Framework for Geospatial Understanding)

Proposer. Construct an executable geospatial problem

Solver. Find the solution to the constructed problem

Remote sensing imagery (I) already encodes rich spatial structure. These can be computed directly, never labeled by hand.



p: program, a: argument, o: output

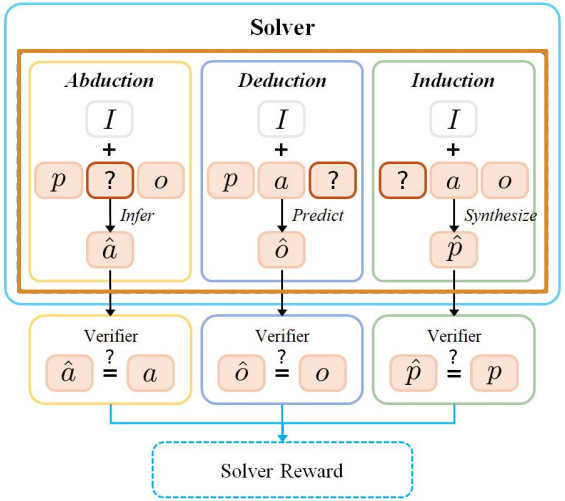
METHOD

Our Approach. GeoX (The Self-play Framework for Geospatial Understanding)

Proposer. Construct an executable geospatial problem

Solver. Find the solution to the constructed problem

One triplet, masked three ways,
affords three distinct modes of reasoning,
engaging a different facet of spatial understanding.



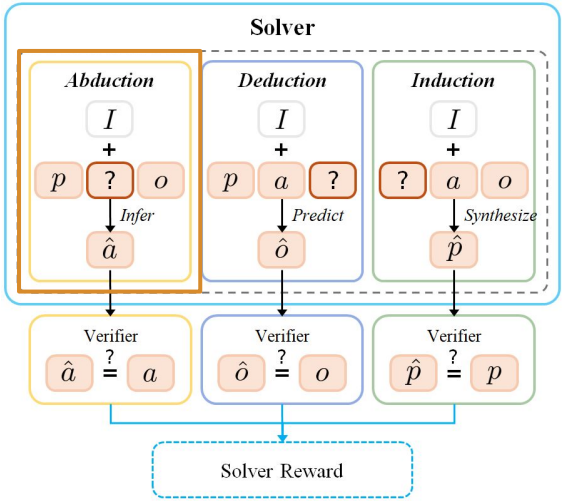
The proposed problem (p, a, o) can be solved in three directions, each training a different kind of reasoning

- Abduction: Solver infers (a) from (p, o).
- Deduction: Solver predicts (o) from (p, a).
- Induction: Solver synthesizes (p) from input-output pairs $\{(a_t, o_t)\}_t$.

Our Approach. GeoX (The Self-play Framework for Geospatial Understanding)

Proposer. Construct an executable geospatial problem

Solver. Find the solution to the constructed problem



The proposed problem (p, a, o) can be solved in three directions, each training a different kind of reasoning

- Abduction: Solver infers (a) from (p, o).
  → reasoning backward from result to cause

Image (I) | program (p) | output (o) | What is the argument (a)?



```
p(I, a):
  import numpy
  import segment
  m = segment(I, a)
  ...
  return o
```

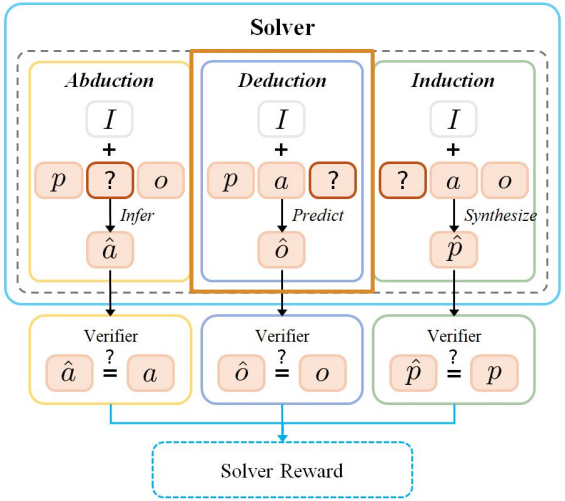
7 → "cargo"

METHOD

Our Approach. GeoX (The Self-play Framework for Geospatial Understanding)

Proposer. Construct an executable geospatial problem

Solver. Find the solution to the constructed problem



The proposed problem (p, a, o) can be solved in three directions, each training a different kind of reasoning

- Deduction: Solver predicts (o) from (p, a).
  → reasoning forward from cause to result

Image (I) | program (p) | output (o) | What is the output (o)?



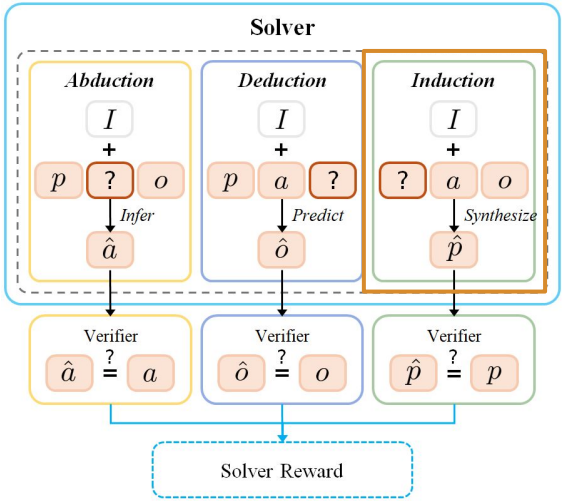
```
p(I, a):
  import numpy
  import segment
  m = segment(I, a)
  ...
  return o
```

"cargo" → 7

Our Approach. GeoX (The Self-play Framework for Geospatial Understanding)

Proposer. Construct an executable geospatial problem

Solver. Find the solution to the constructed problem



The proposed problem (p, a, o) can be solved in three directions, each training a different kind of reasoning

- Induction: Solver synthesizes (p) from input-output pairs $\{(a_t, o_t)\}_t$.
  → generalizing from examples to the rule

Image (I)



input-output pairs $(\{(a_t, o_t)\}_t)$

(cargo, 7)
(fishing, 3)
...

What is the program (p)?

```
p(I, a):
  import numpy
  import segment
  m = segment(I, a)
  ...
  return o
```

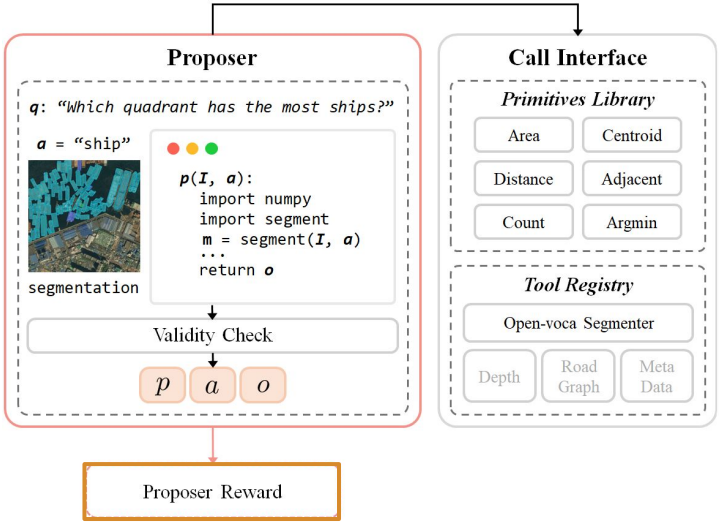
METHOD

Our Approach. GeoX (The Self-play Framework for Geospatial Understanding)

Proposer. Construct an executable geospatial problem (Proposer reward: *learnability*)

Solver. Find the solution to the constructed problem

* R solver trials: R attempts at the same problem.



The *learnability* of a problem is estimated from R solver trials, by averaging their solver rewards.

averaged solver rewards

$$\text{Proposer reward } r^{\text{prop}} = \mathbb{I}\left[\bar{r}^{\text{solve}} > 0\right] \cdot \left(1 - \bar{r}^{\text{solve}}\right)$$

Problem is too hard:

- solver never solves it ($\bar{r}^{\text{solve}} = 0$) ⟶ reward = 0

Problem is too easy:

- solver always solves it ($\bar{r}^{\text{solve}} = 1$) ⟶ reward = 0

Problem is challenging but solvable:

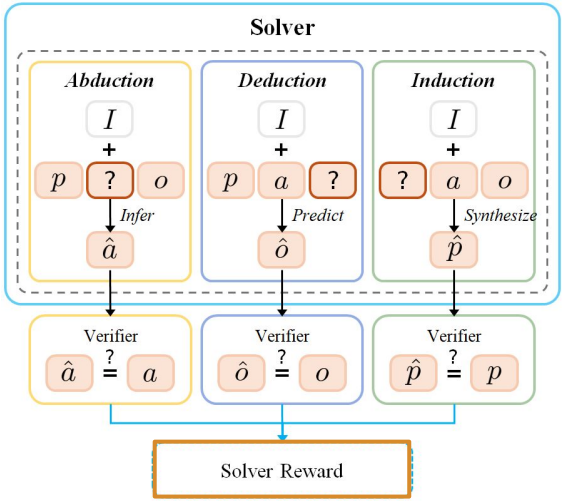
- solver succeed only occasionally (e.g., $\bar{r}^{\text{solve}} = 0.124$) ⟶ reward = 0.875

p: program, a: argument, o: output

Our Approach. GeoX (The Self-play Framework for Geospatial Understanding)

Proposer. Construct an executable geospatial problem

Solver. Find the solution to the constructed problem (Solver reward: *correctness*)



The *correctness* reduces to a simple execution-consistency condition.

Abduction & Induction

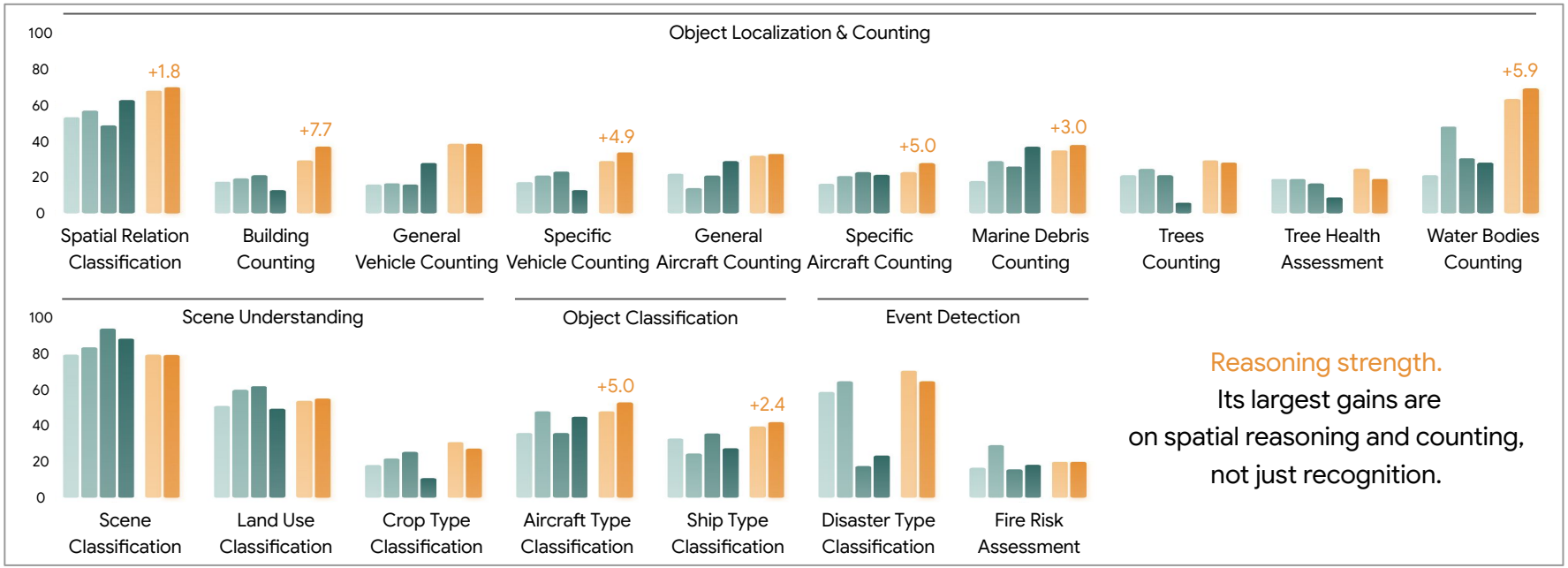
$$r^{\text{solve}}_{\text{abd}}(\hat{a}) = \mathbb{I}\big[p(\hat{a}; I) = o\big], \quad r^{\text{solve}}_{\text{ind}}(\hat{p}) = \frac{1}{|\mathcal{U}|}\sum_{t\in\mathcal{U}} \mathbb{I}\big[\hat{p}(a_t; I) = o_t\big]$$

Deduction

$$r^{\text{solve}}_{\text{ded}}(\hat{o}) = \begin{cases} \max\big(0,\, 1 - |\hat{o} - o| / \max(|o|, 1)\big) & \text{numeric} \\ \mathbb{I}[\text{norm}(\hat{o}) = \text{norm}(o)] & \text{string} \\ \mathbb{I}[\hat{o} = o] & \text{boolean} \\ \text{IoU}(\hat{o}, o) & \text{bounding box} \end{cases}$$

ANALYSIS

Performance Comparison



ANALYSIS

Ablation Study

Base Generator: Drops self-play; problems come from the base model.
Solve Only: Keeps self-play, but rewards only the solver.

Singleton Variants: Keep One Mode

	Abd	Ded	Ind
Object Localization & Counting	35.5	37.2	36.4
Scene Understanding	51.4	52.4	49.4
Object Classification	45.3	46.8	46.3
Event Detection	39.0	42.4	40.3
Average	39.9	41.6	40.3

Drop-one Variants: Remove One Mode

	-Abd	-Ded	-Ind
Object Localization & Counting	36.5	36.4	37.5
Scene Understanding	48.1	50.8	52.1
Object Classification	43.6	49.3	44.3
Event Detection	40.3	43.6	42.8
Average	39.8	41.3	41.5

Training Variants

	Base Generator	Solve Only	Full
Object Localization & Counting	37.4	34.3	39.5
Scene Understanding	51.3	50.1	53.9
Object Classification	42.5	45.3	47.5
Event Detection	46.9	46.6	42.4
Average	41.6	39.8	43.3

ANALYSIS

Ablation Study

Base Generator: Drops self-play; problems come from the base model.
Solve Only: Keeps self-play, but rewards only the solver.

Singleton Variants: Keep One Mode

	Abd	Ded	Ind
Object Localization & Counting	35.5	37.2	36.4
Scene Understanding	51.4	52.4	49.4
Object Classification	45.3	46.8	46.3
Event Detection	39.0	42.4	40.3
Average	39.9	41.6	40.3

Drop-one Variants: Remove One Mode

	-Abd	-Ded	-Ind
Object Localization & Counting	36.5	36.4	37.5
Scene Understanding	48.1	50.8	52.1
Object Classification	43.6	49.3	44.3
Event Detection	40.3	43.6	42.8
Average	39.8	41.3	41.5

Training Variants

	Base Generator	Solve Only	Full
Object Localization & Counting	37.4	34.3	39.5
Scene Understanding	51.3	50.1	53.9
Object Classification	42.5	45.3	47.5
Event Detection	46.9	46.6	42.4
Average	41.6	39.8	43.3

All three modes are needed in different ways.

ANALYSIS

Ablation Study

Base Generator: Drops self-play; problems come from the base model.
Solve Only: Keeps self-play, but rewards only the solver.

Singleton Variants: Keep One Mode

	Abd	Ded	Ind
Object Localization & Counting	35.5	37.2	36.4
Scene Understanding	51.4	52.4	49.4
Object Classification	45.3	46.8	46.3
Event Detection	39.0	42.4	40.3
Average	39.9	41.6	40.3

Drop-one Variants: Remove One Mode

	-Abd	-Ded	-Ind
Object Localization & Counting	36.5	36.4	37.5
Scene Understanding	48.1	50.8	52.1
Object Classification	43.6	49.3	44.3
Event Detection	40.3	43.6	42.8
Average	39.8	41.3	41.5

Training Variants

	Base Generator	Solve Only	Full
Object Localization & Counting	37.4	34.3	39.5
Scene Understanding	51.3	50.1	53.9
Object Classification	42.5	45.3	47.5
Event Detection	46.9	46.6	42.4
Average	41.6	39.8	43.3

The proposer reward is what makes self-play work.

Inside One Deduction Cycle

The program fixes the answers.
The solver's reasoning is verified against it, with no human labels.

TL: Top-Left, TR: Top-Right, BL: Bottom-Left, BR: Bottom-Right

Question (q): " In which quadrant (TL, TR, BL, BR) does the centroid of the largest ship lie in the image? "

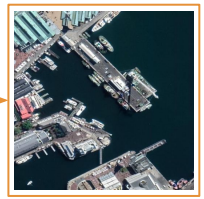
Argument (a): "ship"

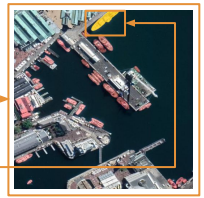
Program (p)

```
def f(image, a):
    import numpy as np
    from scipy.ndimage import center_of_mass
    from tools import segment

    instances = segment(image, a)
    if not isinstance(instances, list) or not instances:
        return "none"
    largest = max(instances, key=lambda m: int(np.sum(m)))
    cy, cx = center_of_mass(largest)
    h, w = largest.shape

    return ("T" if cy < h / 2 else "B") + ("L" if cx < w / 2 else "R")
```







matching (TL, TR, BL, BR)

Output (o): "TR" (program execution)

<think>
1. The input class is "ship".
2. In the image, there are multiple ships docked at the harbor.
3. The task is to find the largest ship and determine the quadrant where its center of mass lies.
4. From the image, the long vessel docked at the top of the image appears to be the largest ship.
5. The center of mass calculation will be applied to this ship to determine the quadrant it falls into.
Let's calculate the center of mass for the largest ship.
The coordinates of the ship in the image are approximately:
- Top-left corner: (260, 60) / Bottom-right corner: (540, 130)
The center of mass calculation for a rectangle is at the midpoint of the top-left and bottom-right corners.
The center of mass for the ship is:
- X-coordinate: (260 + 540) / 2 = 400 / Y-coordinate: (60 + 130) / 2 = 95
Since the center of mass (400, 95) is above the middle of the image (h/2 = 600/2 = 300) and to the right of the middle of the image (w/2 = 600/2 = 300), the quadrant is the top-right (TR). Therefore, the output is 'TR'.
</think>
<answer>TR**</answer>**

Answer(ô): "TR" (our model answer)

ANALYSIS

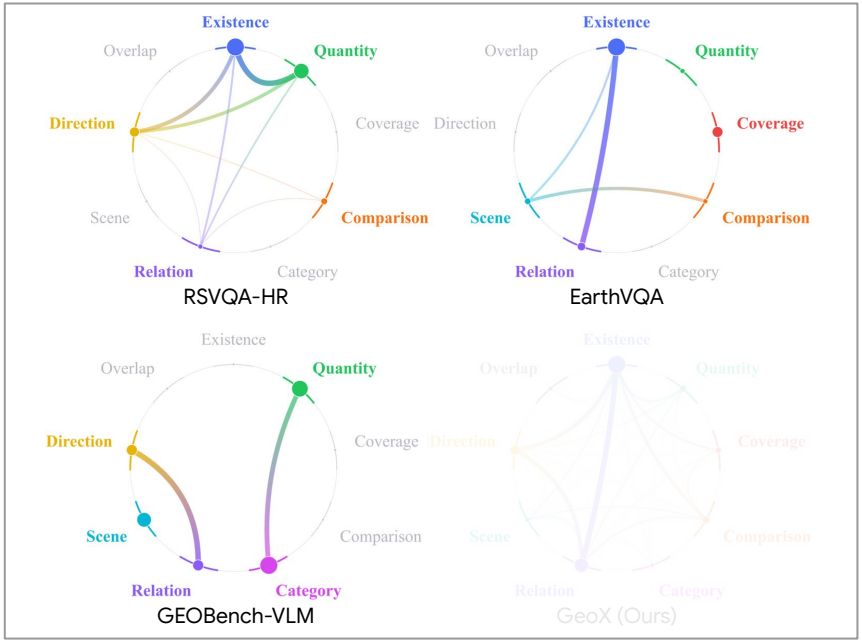
Pairwise Dimensional Coverage

Nine Dimensions: Existence, Quantity, Coverage, Comparison, Category, Relation, Scene, Direction, Overlap

Each problem is tagged with nine dimensions, defined by deterministic rules on its question and program.

Dimension	Definition	Question rule	Code rule
EXISTENCE	Presence of an entity	*are there, is there, any*	`np.any`, `.any()`, `is not None`
QUANTITY	Cardinality of a class	*how many, number of*	`len()`, `np.count_nonzero`
COVERAGE	Pixel area of a region	*area of, ratio of, how large*	`np.sum`, `.sum()`, identifier `area`
COMPARISON	Extremum selection or ordering	*larger/smaller, most/least, X or Y*	`max`, `min`, `argmax`, `argmin`
CATEGORY	Reference to an object class	*what type of object/aircraft*	String literal of an object class (e.g., `"vehicle"`)
RELATION	Continuous spatial proximity	*near, next to, between, adjacent*	`center_of_mass`, `distance_transform`, `cdist`
SCENE	Reference to a scene-type class	*land use, rural/urban, what kind of scene*	String literal of a scene class (e.g., `"water"`)
DIRECTION	Cardinal or quadrant assignment	*north/south/east/west, top of, where is*	String literal in {`north`, `TL`, `NW`, ...}
OVERLAP	Set-theoretic mask interaction	*both, intersection, overlap*	`np.logical_and`, `m & m`, `m | m`

Self-play composes dimensions more freely, pairing them in ways curated benchmarks rarely do.



ANALYSIS

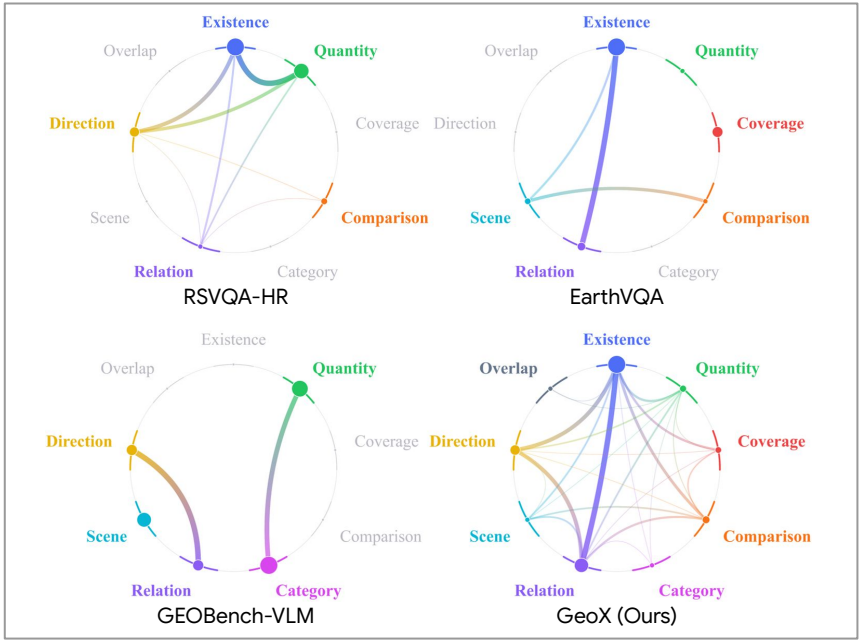
Pairwise Dimensional Coverage

Nine Dimensions: Existence, Quantity, Coverage, Comparison, Category, Relation, Scene, Direction, Overlap

Each problem is tagged with nine dimensions, defined by deterministic rules on its question and program.

Dimension	Definition	Question rule	Code rule
EXISTENCE	Presence of an entity	*are there*, *is there*, *any*	`np.any`, `.any()`, `is not None`
QUANTITY	Cardinality of a class	*how many*, *number of*	`len()`, `np.count_nonzero`
COVERAGE	Pixel area of a region	*area of*, *ratio of*, *how large*	`np.sum`, `.sum()`, identifier `area`
COMPARISON	Extremum selection or ordering	*larger/smaller*, *most/least*, *X or Y*	`max`, `min`, `argmax`, `argmin`
CATEGORY	Reference to an object class	*what type of object/aircraft*	String literal of an object class (e.g., `"vehicle"`)
RELATION	Continuous spatial proximity	*near*, *next to*, *between*, *adjacent*	`center_of_mass`, `distance_transform`, `cdist`
SCENE	Reference to a scene-type class	*land use*, *rural/urban*, *what kind of scene*	String literal of a scene class (e.g., `"water"`)
DIRECTION	Cardinal or quadrant assignment	*north/south/east/west*, *top of*, *where is*	String literal in {`north`, `TL`, `NW`, ...}
OVERLAP	Set-theoretic mask interaction	*both*, *intersection*, *overlap*	`np.logical_and`, `m & m`, `m | m`

Self-play composes dimensions more freely, pairing them in ways curated benchmarks rarely do.



TAKEAWAY

Geospatial Understanding

- We teach a VLM to perform geospatial reasoning on remote sensing imagery by generating problems and solving them, **without any human-curated QA**.
- Validated on three **geospatial VQA benchmarks**, our model matches or beats VLMs fine-tuned on millions of curated pairs, with the largest gains on spatial relations and counting.
- In practice, analysts can ask spatial questions about any scene, such as "*which damaged buildings are farthest from the nearest hospital*" or "*which shelters have no road within reach*", even for questions no one prepared for in advance.