Take-home assignment: Choose one specific generalization problem in the areas of embodied AI or robotics, and finish the following.

1. Identify the SOTA method
2. Replicate the code
3. Propose some new ideas
4. Implement one specific idea

## PART-I: IDENTIFICATION OF SOTA METHOD

For the assignment, the chosen specific generalization is semantic generalization. Semantic Generalization is the ability of a robot to interact with objects it has never seen before. The SOTA method identified is OpenVLA, which is an open-source Vision-Language-Action model trained on real world robot data. OpenVLA is built on 7 billion parameters and consists of Llama2 for LLM and for vision encoders it uses DINOv2 with SigLIP. It is trained on Open X-Embodiment Dataset. OpenVLA takes input of an image with a language instruction and returns a tokenized robot action as output.

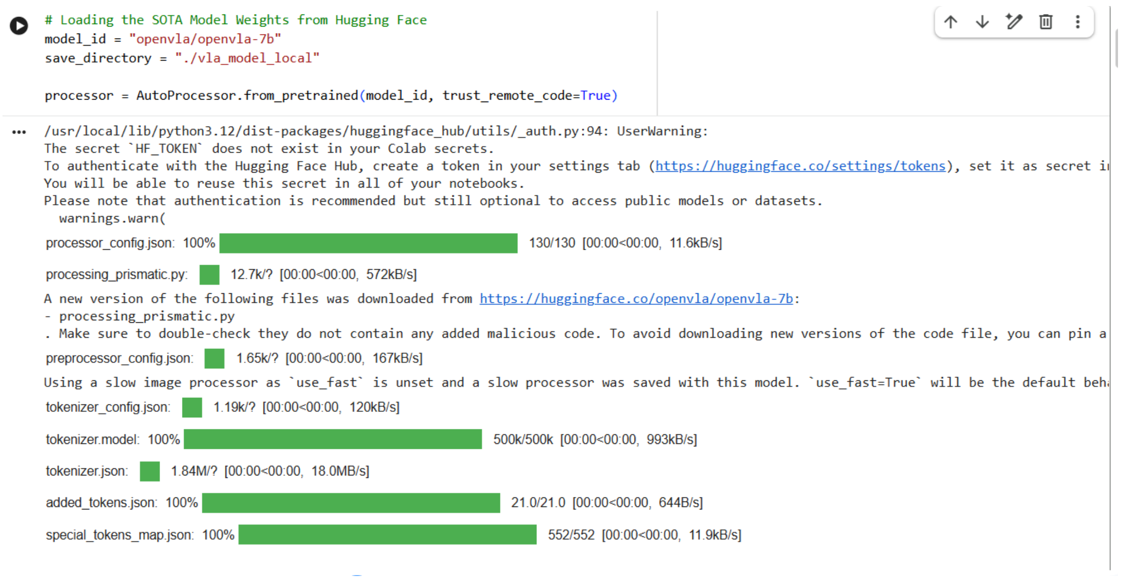
Robot Action Space: It consists of 7 dimensions: [Δposx, Δposy, Δposz, Δrotx, Δroty, Δrotz, Δgripper].

Each dimension is scaled to [-1,1] and discretized to 255 bins (except Δgripper, which is binary 0=close and 1=open).

The success rate for OpenVLA in semantic generalization is 36.3% (visual-87%, motion-60% and physical-76%).

## PART-II: REPLICATION OF OpenVLA CODE

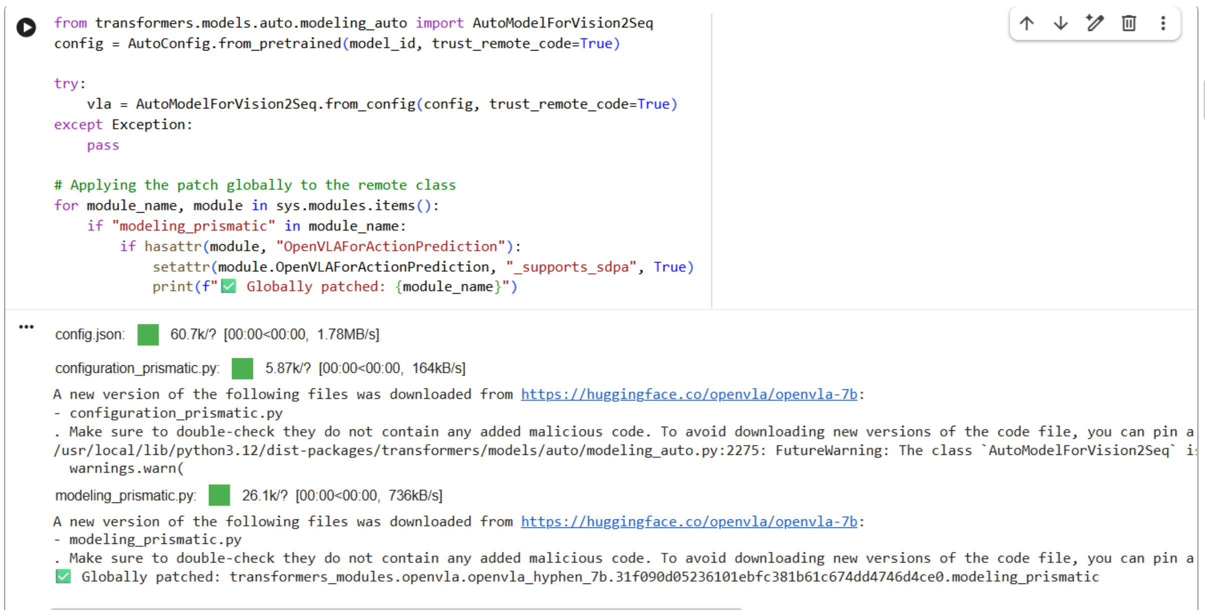
This code initializes the OpenVLA-7b processor by downloading its configuration and multi-modal processing scripts directly from Hugging Face. The trust_remote_code=True flag is required because OpenVLA uses custom logic (Prismatic VLMs) to handle the interaction between vision and language data that is not part of the standard Transformers library.



```python
# Loading the SOTA Model Weights from Hugging Face
model_id = "openvla/openvla-7b"
save_directory = "./vla_model_local"

processor = AutoProcessor.from_pretrained(model_id, trust_remote_code=True)
```

```
/usr/local/lib/python3.12/dist-packages/huggingface_hub/utils/_auth.py:94: UserWarning:
The secret `HF_TOKEN` does not exist in your Colab secrets.
To authenticate with the Hugging Face Hub, create a token in your settings tab (https://huggingface.co/settings/tokens), set it as secret i
You will be able to reuse this secret in all of your notebooks.
Please note that authentication is recommended but still optional to access public models or datasets.
  warnings.warn(
processor_config.json: 100% 130/130 [00:00<00:00, 11.6kB/s]
processing_prismatic.py:  12.7k/? [00:00<00:00, 572kB/s]
A new version of the following files was downloaded from https://huggingface.co/openvla/openvla-7b:
- processing_prismatic.py
. Make sure to double-check they do not contain any added malicious code. To avoid downloading new versions of the code file, you can pin a
preprocessor_config.json:  1.65k/? [00:00<00:00, 167kB/s]
Using a slow image processor as `use_fast` is unset and a slow processor was saved with this model. `use_fast=True` will be the default beh
tokenizer_config.json:  1.19k/? [00:00<00:00, 120kB/s]
tokenizer.model: 100% 500k/500k [00:00<00:00, 993kB/s]
tokenizer.json:  1.84M/? [00:00<00:00, 18.0MB/s]
added_tokens.json: 100% 21.0/21.0 [00:00<00:00, 644B/s]
special_tokens_map.json: 100% 552/552 [00:00<00:00, 11.9kB/s]
```

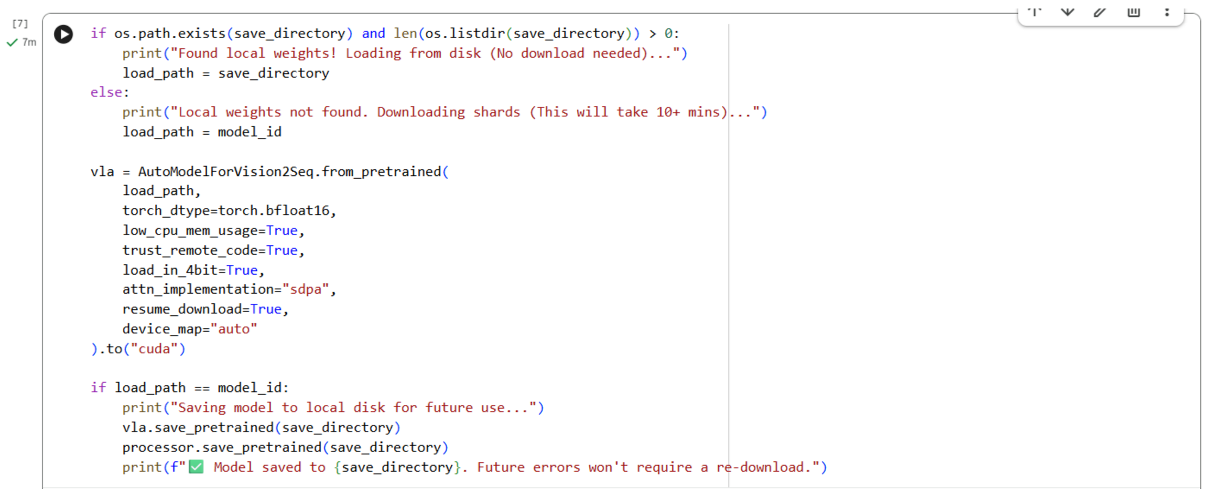
This code performs an import to fix a compatibility bug in the OpenVLA architecture before the actual model is loaded.



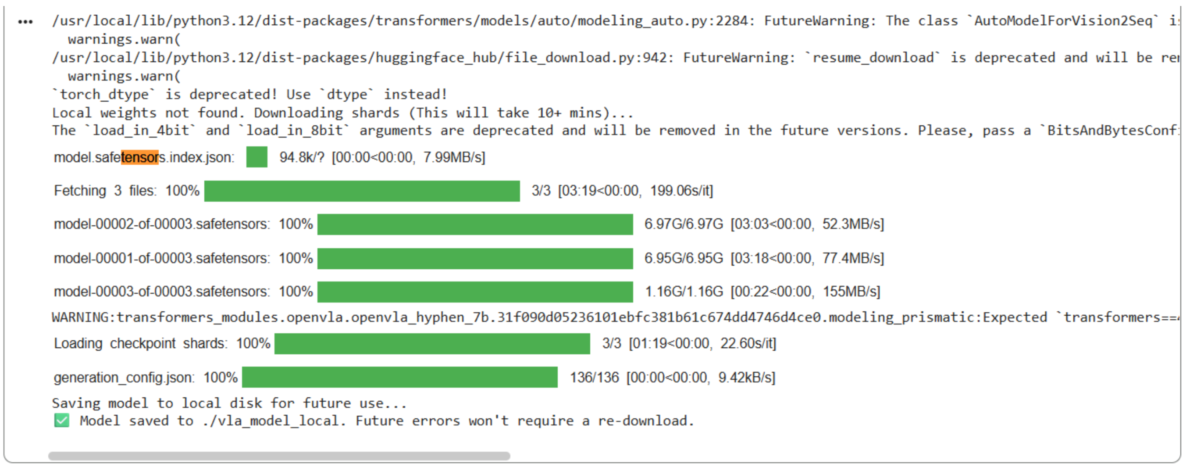
```
from transformers.models.auto.modeling_auto import AutoModelForVision2Seq
config = AutoConfig.from_pretrained(model_id, trust_remote_code=True)

try:
    vla = AutoModelForVision2Seq.from_config(config, trust_remote_code=True)
except Exception:
    pass

# Applying the patch globally to the remote class
for module_name, module in sys.modules.items():
    if "modeling_prismatic" in module_name:
        if hasattr(module, "OpenVLAForActionPrediction"):
            setattr(module.OpenVLAForActionPrediction, "_supports_sdpa", True)
            print(f"✅ Globally patched: {module_name}")
```

```
config.json: 🟩 60.7k/? [00:00<00:00, 1.78MB/s]
configuration_prismatic.py: 🟩 5.87k/? [00:00<00:00, 164kB/s]
A new version of the following files was downloaded from https://huggingface.co/openvla/openvla-7b:
- configuration_prismatic.py
. Make sure to double-check they do not contain any added malicious code. To avoid downloading new versions of the code file, you can pin a
/usr/local/lib/python3.12/dist-packages/transformers/models/auto/modeling_auto.py:2275: FutureWarning: The class `AutoModelForVision2Seq` i
  warnings.warn(
modeling_prismatic.py: 🟩 26.1k/? [00:00<00:00, 736kB/s]
A new version of the following files was downloaded from https://huggingface.co/openvla/openvla-7b:
- modeling_prismatic.py
. Make sure to double-check they do not contain any added malicious code. To avoid downloading new versions of the code file, you can pin a
✅ Globally patched: transformers_modules.openvla.openvla_hyphen_7b.31f090d05236101ebfc381b61c674dd4746d4ce0.modeling_prismatic
```

This code implements a loading system that checks for local weights to avoid redundant 7B-parameter downloads while optimizing memory with 4-bit quantization. If the model is downloaded for the first time, it automatically saves a copy to local disk, ensuring that future sessions can bypass the lengthy download wait.

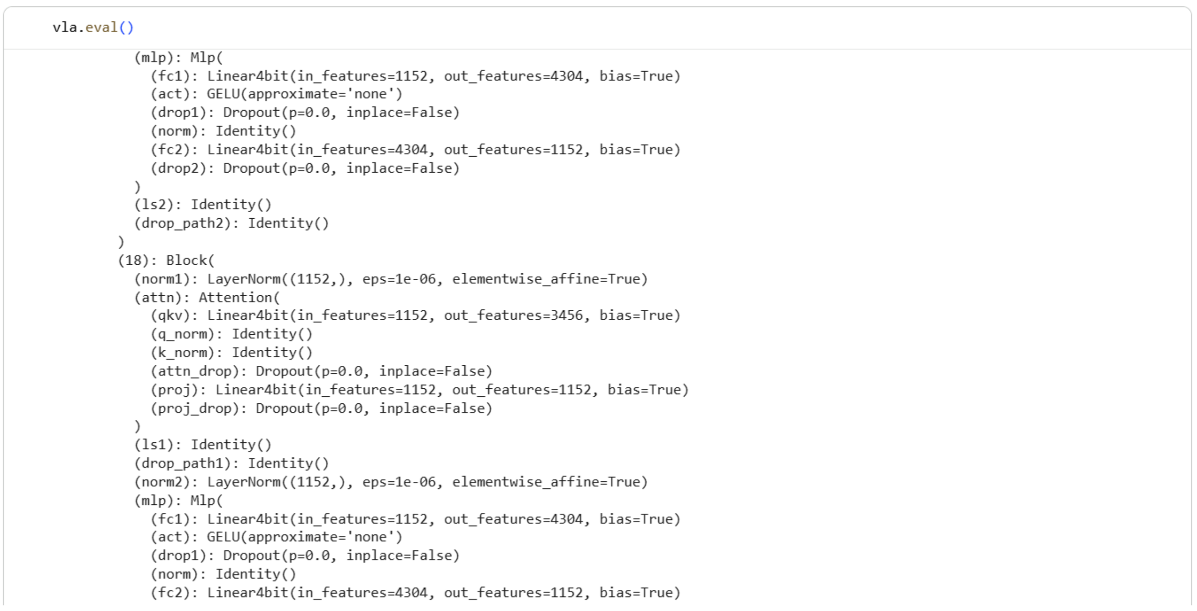


```
[7]
✓ 7m
if os.path.exists(save_directory) and len(os.listdir(save_directory)) > 0:
    print("Found local weights! Loading from disk (No download needed)...")
    load_path = save_directory
else:
    print("Local weights not found. Downloading shards (This will take 10+ mins)...")
    load_path = model_id

vla = AutoModelForVision2Seq.from_pretrained(
    load_path,
    torch_dtype=torch.bfloat16,
    low_cpu_mem_usage=True,
    trust_remote_code=True,
    load_in_4bit=True,
    attn_implementation="sdpa",
    resume_download=True,
    device_map="auto"
).to("cuda")

if load_path == model_id:
    print("Saving model to local disk for future use...")
    vla.save_pretrained(save_directory)
    processor.save_pretrained(save_directory)
    print(f"✅ Model saved to {save_directory}. Future errors won't require a re-download.")
```



```
/usr/local/lib/python3.12/dist-packages/transformers/models/auto/modeling_auto.py:2284: FutureWarning: The class `AutoModelForVision2Seq` i
  warnings.warn(
/usr/local/lib/python3.12/dist-packages/huggingface_hub/file_download.py:942: FutureWarning: `resume_download` is deprecated and will be re
  warnings.warn(
`torch_dtype` is deprecated! Use `dtype` instead!
Local weights not found. Downloading shards (This will take 10+ mins)...
The `load_in_4bit` and `load_in_8bit` arguments are deprecated and will be removed in the future versions. Please, pass a `BitsAndBytesConf
model.safetensors.index.json:  94.8k/? [00:00<00:00, 7.99MB/s]
Fetching 3 files: 100%  3/3 [03:19<00:00, 199.06s/it]
model-00002-of-00003.safetensors: 100%  6.97G/6.97G [03:03<00:00, 52.3MB/s]
model-00001-of-00003.safetensors: 100%  6.95G/6.95G [03:18<00:00, 77.4MB/s]
model-00003-of-00003.safetensors: 100%  1.16G/1.16G [00:22<00:00, 155MB/s]
WARNING:transformers_modules.openvla.openvla_hyphen_7b.31f090d05236101ebfc381b61c674dd4746d4ce0.modeling_prismatic:Expected `transformers==
Loading checkpoint shards: 100%  3/3 [01:19<00:00, 22.60s/it]
generation_config.json: 100%  136/136 [00:00<00:00, 9.42kB/s]
Saving model to local disk for future use...
✅ Model saved to ./vla_model_local. Future errors won't require a re-download.
```

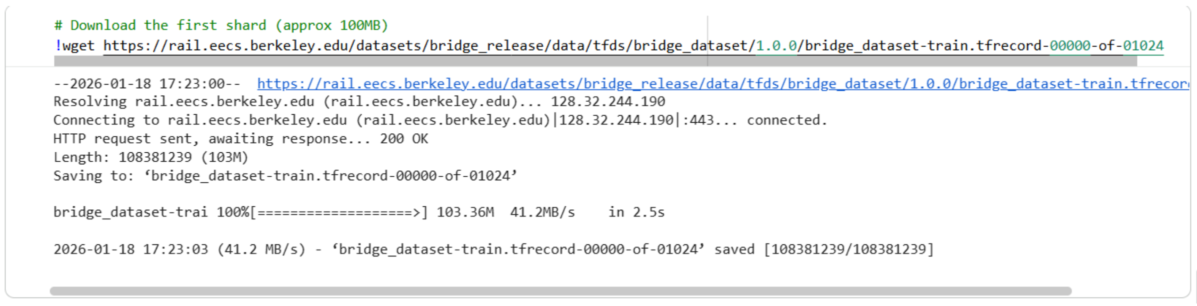
This command sets the model to evaluation mode.



```
vla.eval()
```

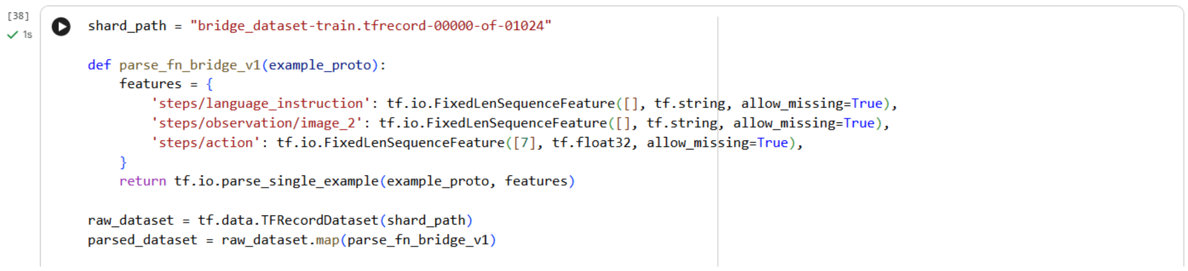
```
          (mlp): Mlp(
            (fc1): Linear4bit(in_features=1152, out_features=4304, bias=True)
            (act): GELU(approximate='none')
            (drop1): Dropout(p=0.0, inplace=False)
            (norm): Identity()
            (fc2): Linear4bit(in_features=4304, out_features=1152, bias=True)
            (drop2): Dropout(p=0.0, inplace=False)
          )
          (ls2): Identity()
          (drop_path2): Identity()
        )
        (18): Block(
          (norm1): LayerNorm((1152,), eps=1e-06, elementwise_affine=True)
          (attn): Attention(
            (qkv): Linear4bit(in_features=1152, out_features=3456, bias=True)
            (q_norm): Identity()
            (k_norm): Identity()
            (attn_drop): Dropout(p=0.0, inplace=False)
            (proj): Linear4bit(in_features=1152, out_features=1152, bias=True)
            (proj_drop): Dropout(p=0.0, inplace=False)
          )
          (ls1): Identity()
          (drop_path1): Identity()
          (norm2): LayerNorm((1152,), eps=1e-06, elementwise_affine=True)
          (mlp): Mlp(
            (fc1): Linear4bit(in_features=1152, out_features=4304, bias=True)
            (act): GELU(approximate='none')
            (drop1): Dropout(p=0.0, inplace=False)
            (norm): Identity()
            (fc2): Linear4bit(in_features=4304, out_features=1152, bias=True)
```

This code defines a parsing function to decode the RLDS (Reinforcement Learning Datasets) format from raw TFRecord files, specifically tailored for the Bridge V2 robot dataset. It extracts key sequential data: camera images, natural language instructions, and 7-dimensional action vectors to map them into a structured TensorFlow dataset ready for model consumption.



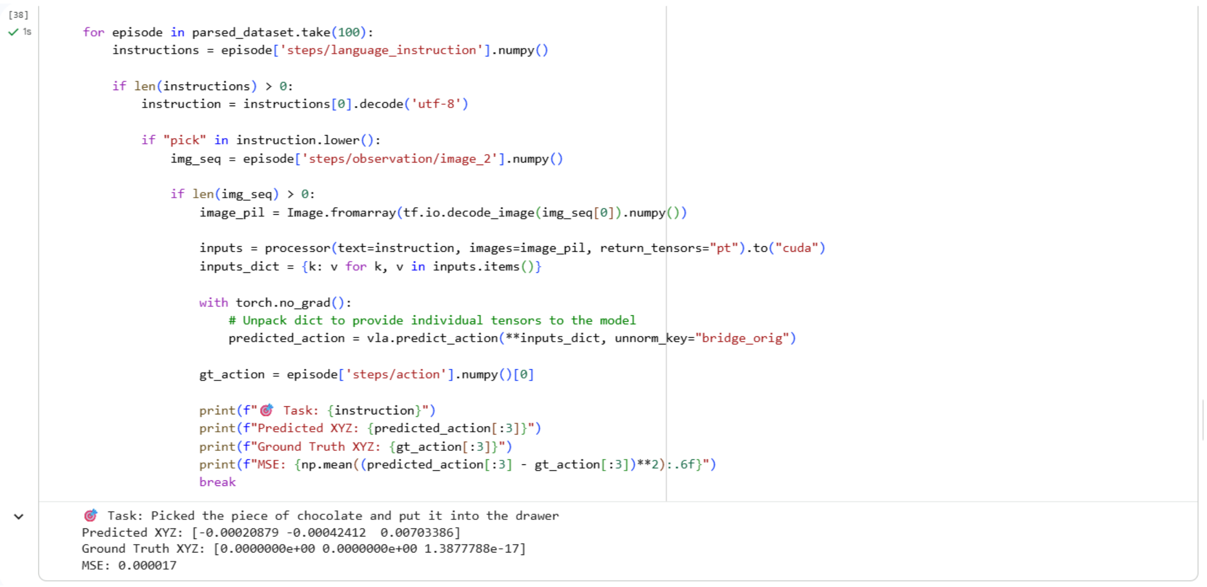
```
# Download the first shard (approx 100MB)
!wget https://rail.eecs.berkeley.edu/datasets/bridge_release/data/tfds/bridge_dataset/1.0.0/bridge_dataset-train.tfrecord-00000-of-01024
```

```
--2026-01-18 17:23:00--  https://rail.eecs.berkeley.edu/datasets/bridge_release/data/tfds/bridge_dataset/1.0.0/bridge_dataset-train.tfrecor
Resolving rail.eecs.berkeley.edu (rail.eecs.berkeley.edu)... 128.32.244.190
Connecting to rail.eecs.berkeley.edu (rail.eecs.berkeley.edu)|128.32.244.190|:443... connected.
HTTP request sent, awaiting response... 200 OK
Length: 108381239 (103M)
Saving to: ‘bridge_dataset-train.tfrecord-00000-of-01024’

bridge_dataset-trai 100%[===================>] 103.36M  41.2MB/s    in 2.5s

2026-01-18 17:23:03 (41.2 MB/s) - ‘bridge_dataset-train.tfrecord-00000-of-01024’ saved [108381239/108381239]
```



```python
shard_path = "bridge_dataset-train.tfrecord-00000-of-01024"

def parse_fn_bridge_v1(example_proto):
    features = {
        'steps/language_instruction': tf.io.FixedLenSequenceFeature([], tf.string, allow_missing=True),
        'steps/observation/image_2': tf.io.FixedLenSequenceFeature([], tf.string, allow_missing=True),
        'steps/action': tf.io.FixedLenSequenceFeature([7], tf.float32, allow_missing=True),
    }
    return tf.io.parse_single_example(example_proto, features)

raw_dataset = tf.data.TFRecordDataset(shard_path)
parsed_dataset = raw_dataset.map(parse_fn_bridge_v1)
```

This code iterates through the Bridge V1 dataset to find "pick" tasks, decodes the raw image sequence, and runs inference using the OpenVLA model. It then compares the model's predicted 7D action vector against the dataset's ground truth and calculates the Mean Squared Error (MSE) to evaluate the accuracy of the robot's predicted spatial movement.



```python
for episode in parsed_dataset.take(100):
    instructions = episode['steps/language_instruction'].numpy()

    if len(instructions) > 0:
        instruction = instructions[0].decode('utf-8')

        if "pick" in instruction.lower():
            img_seq = episode['steps/observation/image_2'].numpy()

            if len(img_seq) > 0:
                image_pil = Image.fromarray(tf.io.decode_image(img_seq[0]).numpy())

                inputs = processor(text=instruction, images=image_pil, return_tensors="pt").to("cuda")
                inputs_dict = {k: v for k, v in inputs.items()}

                with torch.no_grad():
                    # Unpack dict to provide individual tensors to the model
                    predicted_action = vla.predict_action(**inputs_dict, unnorm_key="bridge_orig")

                gt_action = episode['steps/action'].numpy()[0]

                print(f"🎯 Task: {instruction}")
                print(f"Predicted XYZ: {predicted_action[:3]}")
                print(f"Ground Truth XYZ: {gt_action[:3]}")
                print(f"MSE: {np.mean((predicted_action[:3] - gt_action[:3])**2):.6f}")
                break
```

```
🎯 Task: Picked the piece of chocolate and put it into the drawer
Predicted XYZ: [-0.00020879 -0.00042412  0.00703386]
Ground Truth XYZ: [0.0000000e+00 0.0000000e+00 1.3877788e-17]
MSE: 0.000017
```

PART-III: PROPOSING SOME IDEAS FOR SEMANTIC GENERALIZATION

### Idea A: Semantic Descriptor Prompting

Instead of a simple label, provide the model with a physical description of the unseen object.

- The Concept: Use a "Teacher" VLM (like a larger Gemini or GPT-4 model) to generate a description: "The object is a blue, squishy toy shaped like a star."
- The VLA Integration: Prepend this description to the instruction. "Task: Pick up the star-shaped toy. Description: It is blue and squishy. Action:"
- Expected Improvement: It bridges the gap between the unseen visual pixels and the model's pre-trained language concepts, helping the model "recognize" the object through words it already knows.

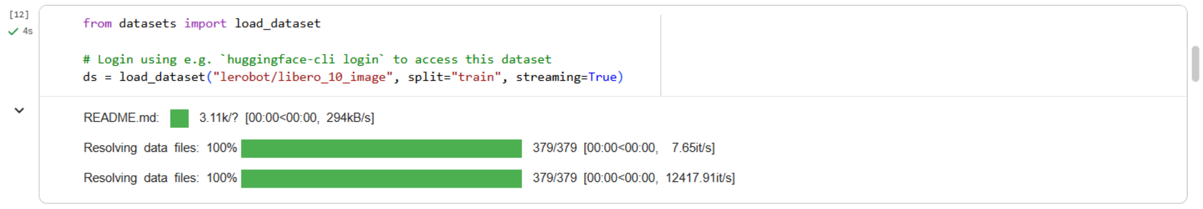
### Idea B: Visual Foundation Guidance (Visual Prompting)

Unseen objects can sometimes cause the VLA's attention to "smear" across the background because it does not have a strong visual anchor for that specific item.

- The Concept: Use a model like SAM (Segment Anything Model) to identify the specific object the user mentions in the instruction.
- The VLA Integration: Instead of just the raw image, we provide the image with a visual prompt. Like a red bounding box, a highlighted mask, or two-panel visual guidance around the object of interest.
- Expected Improvement**:** It removes the "Where is it?" search difficulty, allowing the VLA to focus entirely on the "How do I move toward it?" task.

## PART-IV: IMPLEMENTATION OF VISUAL GUIDANCE

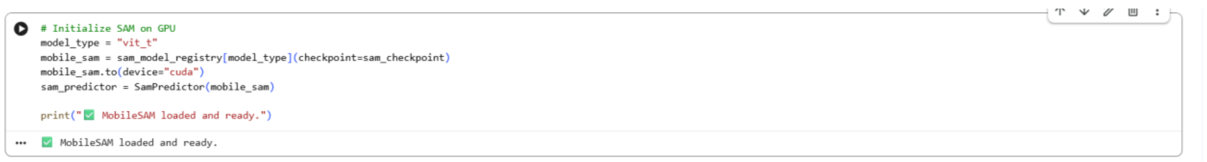
This code uses the Hugging Face datasets library to initialize a streaming iterator for the libero_10_image robot dataset.



```
[12]
✓ 4s
from datasets import load_dataset

# Login using e.g. `huggingface-cli login` to access this dataset
ds = load_dataset("lerobot/libero_10_image", split="train", streaming=True)
```

```
README.md: 🟩 3.11k/? [00:00<00:00, 294kB/s]
Resolving data files: 100% 379/379 [00:00<00:00,  7.65it/s]
Resolving data files: 100% 379/379 [00:00<00:00, 12417.91it/s]
```

This code initializes MobileSAM, a lightweight version of the Segment Anything Model, by loading its vision transformer weights (vit_t) onto the GPU. It then wraps the model in a SamPredictor class, enabling fast, real-time object segmentation and mask generation for robot's camera observations.

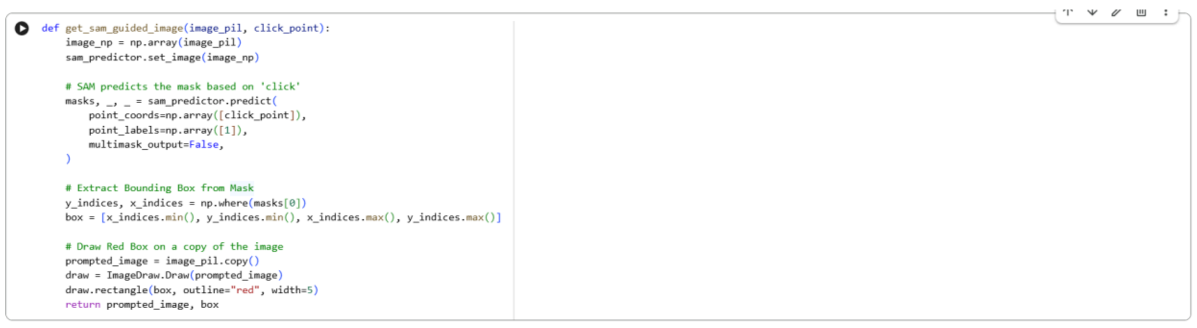


```
# Initialize SAM on GPU
model_type = "vit_t"
mobile_sam = sam_model_registry[model_type](checkpoint=sam_checkpoint)
mobile_sam.to(device="cuda")
sam_predictor = SamPredictor(mobile_sam)

print("✅ MobileSAM loaded and ready.")
```

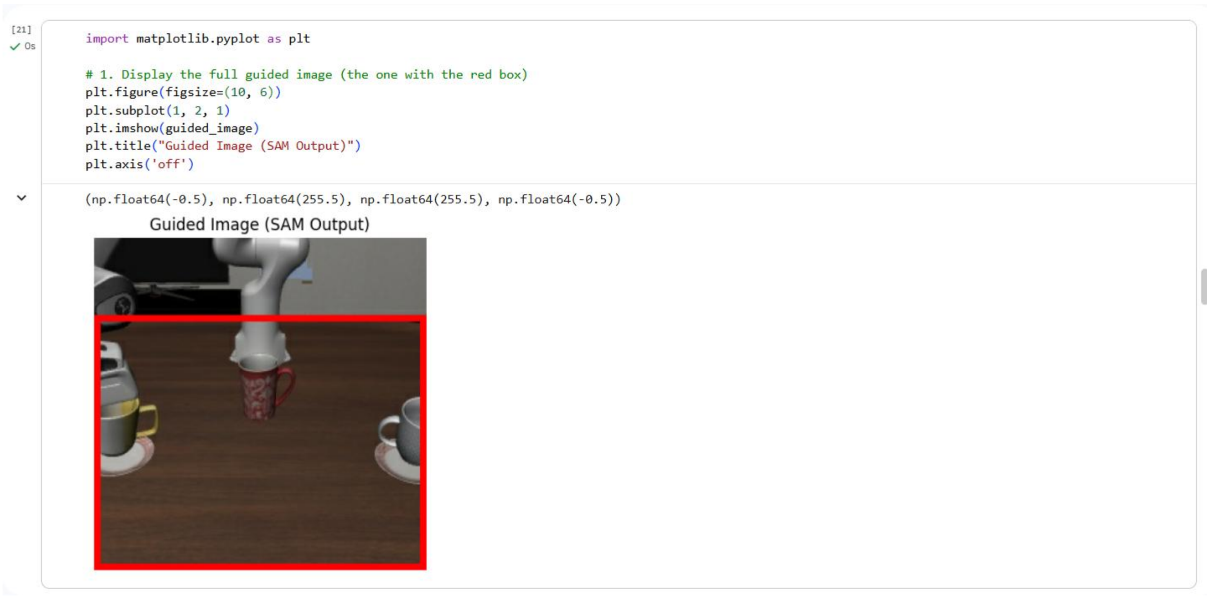
```
✅ MobileSAM loaded and ready.
```

This function uses MobileSAM to generate a visual prompt for the model by converting a single pixel "click" into a precise bounding box. It segments the object at the coordinates, calculates its spatial boundaries, and returns a modified image with a red rectangle highlighting the target, which helps the VLA model focus on the specific object to be manipulated.

### Implementation 1: Red Bounding Box



```
def get_sam_guided_image(image_pil, click_point):
    image_np = np.array(image_pil)
    sam_predictor.set_image(image_np)

    # SAM predicts the mask based on 'click'
    masks, _, _ = sam_predictor.predict(
        point_coords=np.array([click_point]),
        point_labels=np.array([1]),
        multimask_output=False,
    )

    # Extract Bounding Box from Mask
    y_indices, x_indices = np.where(masks[0])
    box = [x_indices.min(), y_indices.min(), x_indices.max(), y_indices.max()]

    # Draw Red Box on a copy of the image
    prompted_image = image_pil.copy()
    draw = ImageDraw.Draw(prompted_image)
    draw.rectangle(box, outline="red", width=5)
    return prompted_image, box
```

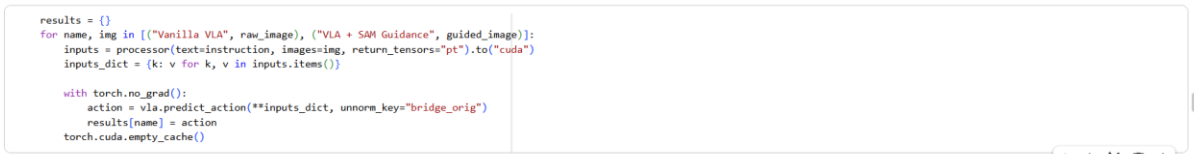


```
[21]
```

```python
import matplotlib.pyplot as plt

# 1. Display the full guided image (the one with the red box)
plt.figure(figsize=(10, 6))
plt.subplot(1, 2, 1)
plt.imshow(guided_image)
plt.title("Guided Image (SAM Output)")
plt.axis('off')
```

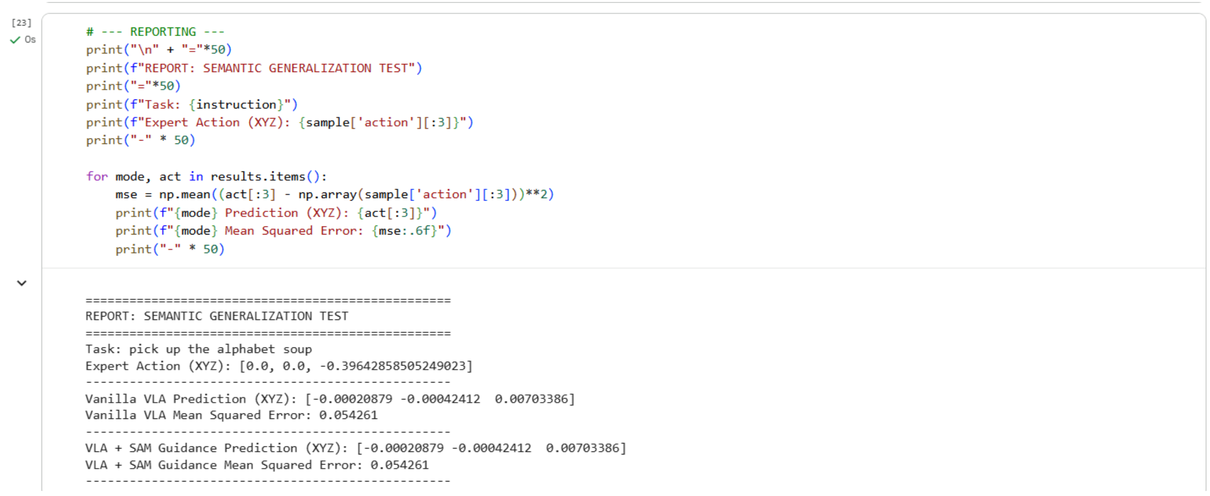
```
(np.float64(-0.5), np.float64(255.5), np.float64(255.5), np.float64(-0.5))
```

This code performs a comparative analysis of robot action predictions by running the OpenVLA model on two versions of the same observation: a standard ("Vanilla") image and one augmented with SAM-generated visual guidance.



```python
results = {}
for name, img in [("Vanilla VLA", raw_image), ("VLA + SAM Guidance", guided_image)]:
    inputs = processor(text=instruction, images=img, return_tensors="pt").to("cuda")
    inputs_dict = {k: v for k, v in inputs.items()}

    with torch.no_grad():
        action = vla.predict_action(**inputs_dict, unnorm_key="bridge_orig")
        results[name] = action
    torch.cuda.empty_cache()
```

This code generates a performance report for a Semantic Generalization Test, comparing the precision of the robot's predicted movements against the "Expert Action" (ground truth) from the dataset.



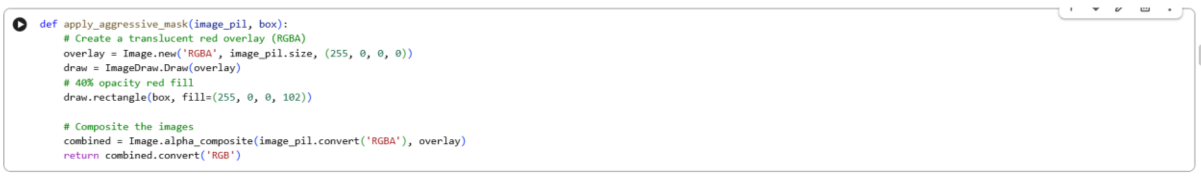
```
[23]
```

```python
# --- REPORTING ---
print("\n" + "="*50)
print(f"REPORT: SEMANTIC GENERALIZATION TEST")
print("="*50)
print(f"Task: {instruction}")
print(f"Expert Action (XYZ): {sample['action'][:3]}")
print("-" * 50)

for mode, act in results.items():
    mse = np.mean((act[:3] - np.array(sample['action'][:3]))**2)
    print(f"{mode} Prediction (XYZ): {act[:3]}")
    print(f"{mode} Mean Squared Error: {mse:.6f}")
    print("-" * 50)
```

```

==================================================
REPORT: SEMANTIC GENERALIZATION TEST
==================================================
Task: pick up the alphabet soup
Expert Action (XYZ): [0.0, 0.0, -0.39642858505249023]
--------------------------------------------------
Vanilla VLA Prediction (XYZ): [-0.00020879 -0.00042412  0.00703386]
Vanilla VLA Mean Squared Error: 0.054261
--------------------------------------------------
VLA + SAM Guidance Prediction (XYZ): [-0.00020879 -0.00042412  0.00703386]
VLA + SAM Guidance Mean Squared Error: 0.054261
--------------------------------------------------
```

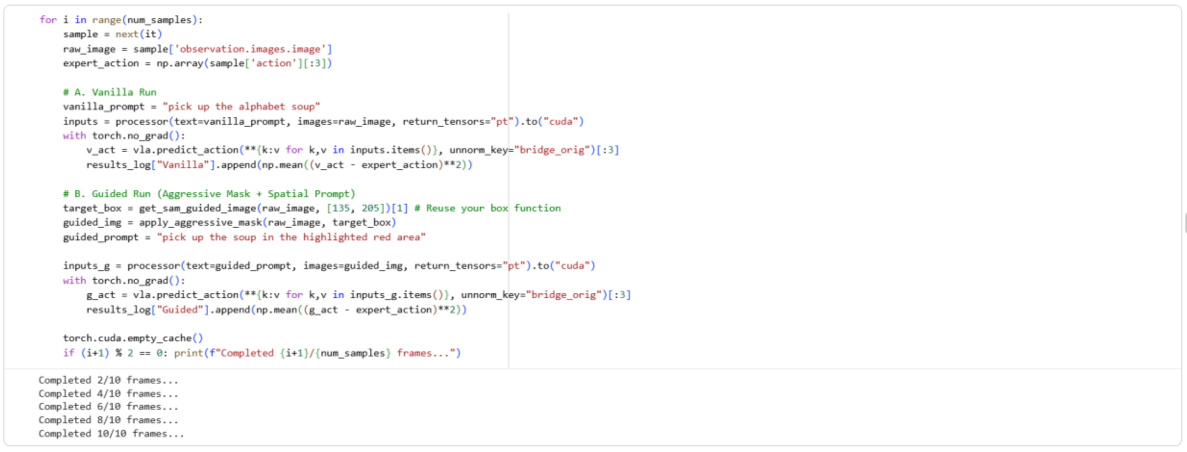
## Implementation 2: Highlighted Mask

This function creates a high-contrast visual prompt by applying a translucent red highlight (40% opacity) directly over the target object's bounding box.



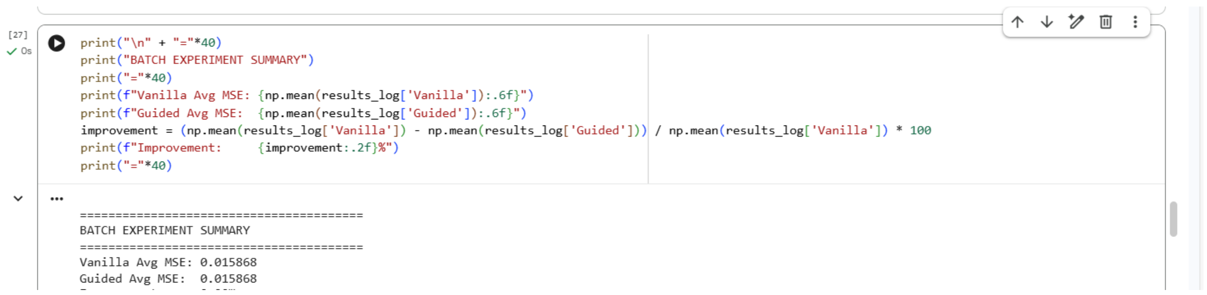
```python
def apply_aggressive_mask(image_pil, box):
    # Create a translucent red overlay (RGBA)
    overlay = Image.new('RGBA', image_pil.size, (255, 0, 0, 0))
    draw = ImageDraw.Draw(overlay)
    # 40% opacity red fill
    draw.rectangle(box, fill=(255, 0, 0, 102))

    # Composite the images
    combined = Image.alpha_composite(image_pil.convert('RGBA'), overlay)
    return combined.convert('RGB')
```

This code runs a comparative evaluation loop over multiple dataset samples to test if visual "guidance" improves the robot's accuracy. For each frame, it calculates the Mean Squared Error (MSE) for a "Vanilla" run using only text and a "Guided" run that combines a modified image (aggressive red mask) with a spatially-aware text prompt.



```
for i in range(num_samples):
    sample = next(it)
    raw_image = sample['observation.images.image']
    expert_action = np.array(sample['action'][:3])

    # A. Vanilla Run
    vanilla_prompt = "pick up the alphabet soup"
    inputs = processor(text=vanilla_prompt, images=raw_image, return_tensors="pt").to("cuda")
    with torch.no_grad():
        v_act = vla.predict_action(**{k:v for k,v in inputs.items()}, unnorm_key="bridge_orig")[:3]
        results_log["Vanilla"].append(np.mean((v_act - expert_action)**2))

    # B. Guided Run (Aggressive Mask + Spatial Prompt)
    target_box = get_sam_guided_image(raw_image, [135, 205])[1] # Reuse your box function
    guided_img = apply_aggressive_mask(raw_image, target_box)
    guided_prompt = "pick up the soup in the highlighted red area"

    inputs_g = processor(text=guided_prompt, images=guided_img, return_tensors="pt").to("cuda")
    with torch.no_grad():
        g_act = vla.predict_action(**{k:v for k,v in inputs_g.items()}, unnorm_key="bridge_orig")[:3]
        results_log["Guided"].append(np.mean((g_act - expert_action)**2))

    torch.cuda.empty_cache()
    if (i+1) % 2 == 0: print(f"Completed {i+1}/{num_samples} frames...")
```

```
Completed 2/10 frames...
Completed 4/10 frames...
Completed 6/10 frames...
Completed 8/10 frames...
Completed 10/10 frames...
```

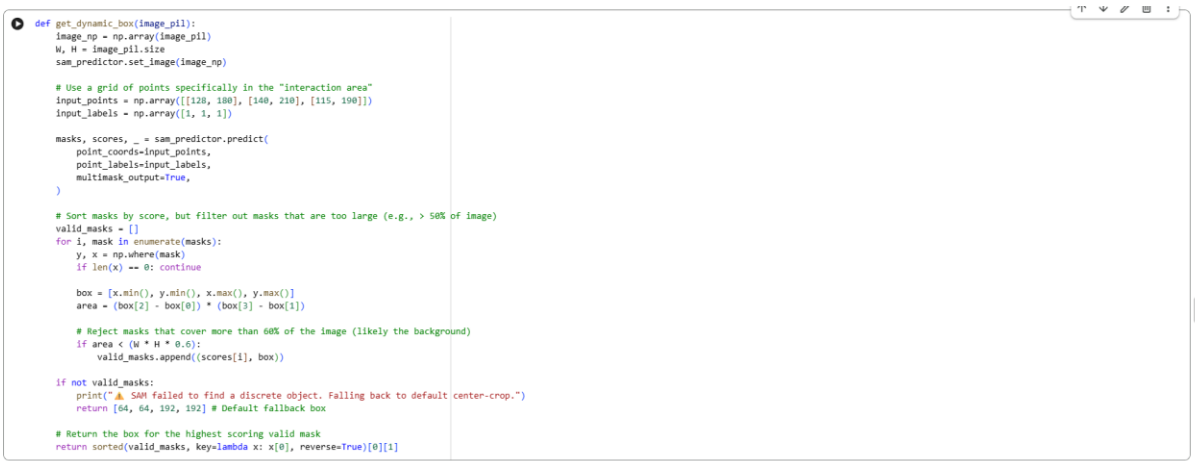


```
[27]
✓ 0s
print("\n" + "="*40)
print("BATCH EXPERIMENT SUMMARY")
print("="*40)
print(f"Vanilla Avg MSE: {np.mean(results_log['Vanilla']):.6f}")
print(f"Guided Avg MSE:  {np.mean(results_log['Guided']):.6f}")
improvement = (np.mean(results_log['Vanilla']) - np.mean(results_log['Guided'])) / np.mean(results_log['Vanilla']) * 100
print(f"Improvement:     {improvement:.2f}%")
print("="*40)
```

```
========================================
BATCH EXPERIMENT SUMMARY
========================================
Vanilla Avg MSE:  0.015868
Guided Avg MSE:   0.015868
```

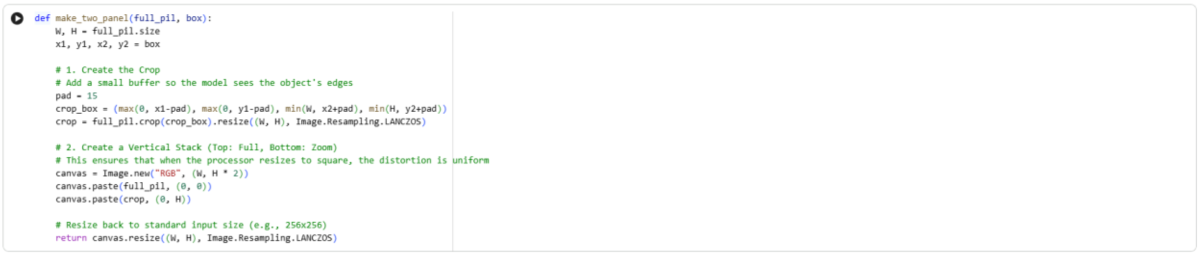
## Implementation 3: Two-Panel Visual Guidance

This code implements a robust object detection strategy by using a grid of multiple "interaction points" rather than a single click to guide MobileSAM.

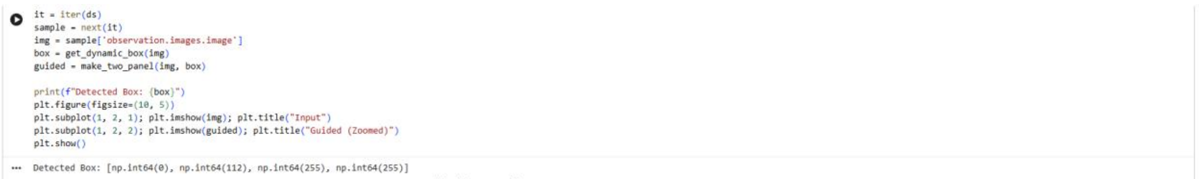


```
def get_dynamic_box(image_pil):
    image_np = np.array(image_pil)
    W, H = image_pil.size
    sam_predictor.set_image(image_np)

    # Use a grid of points specifically in the "interaction area"
    input_points = np.array([[128, 180], [140, 210], [115, 190]])
    input_labels = np.array([1, 1, 1])

    masks, scores, _ = sam_predictor.predict(
        point_coords=input_points,
        point_labels=input_labels,
        multimask_output=True,
    )

    # Sort masks by score, but filter out masks that are too large (e.g., > 50% of image)
    valid_masks = []
    for i, mask in enumerate(masks):
        y, x = np.where(mask)
        if len(x) == 0: continue

        box = [x.min(), y.min(), x.max(), y.max()]
        area = (box[2] - box[0]) * (box[3] - box[1])

        # Reject masks that cover more than 60% of the image (likely the background)
        if area < (W * H * 0.6):
            valid_masks.append((scores[i], box))

    if not valid_masks:
        print("⚠️ SAM failed to find a discrete object. Falling back to default center-crop.")
        return [64, 64, 192, 192] # Default fallback box

    # Return the box for the highest scoring valid mask
    return sorted(valid_masks, key=lambda x: x[0], reverse=True)[0][1]
```

This code generates a "two-panel" input image by vertically stacking the original full-frame view on top of a zoomed-in, high-resolution crop of the target object. By resizing this combined canvas back to a standard input size, it allows the OpenVLA model to simultaneously process global spatial context and fine-grained object details without losing visual information.

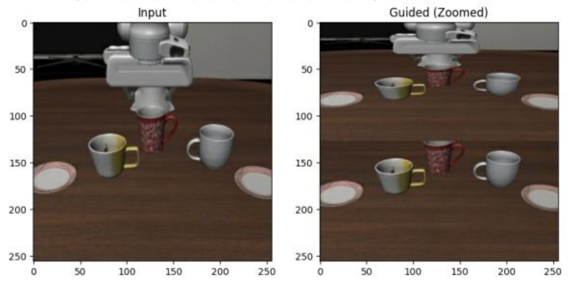


```python
def make_two_panel(full_pil, box):
    W, H = full_pil.size
    x1, y1, x2, y2 = box

    # 1. Create the Crop
    # Add a small buffer so the model sees the object's edges
    pad = 15
    crop_box = (max(0, x1-pad), max(0, y1-pad), min(W, x2+pad), min(H, y2+pad))
    crop = full_pil.crop(crop_box).resize((W, H), Image.Resampling.LANCZOS)

    # 2. Create a Vertical Stack (Top: Full, Bottom: Zoom)
    # This ensures that when the processor resizes to square, the distortion is uniform
    canvas = Image.new("RGB", (W, H * 2))
    canvas.paste(full_pil, (0, 0))
    canvas.paste(crop, (0, H))

    # Resize back to standard input size (e.g., 256x256)
    return canvas.resize((W, H), Image.Resampling.LANCZOS)
```

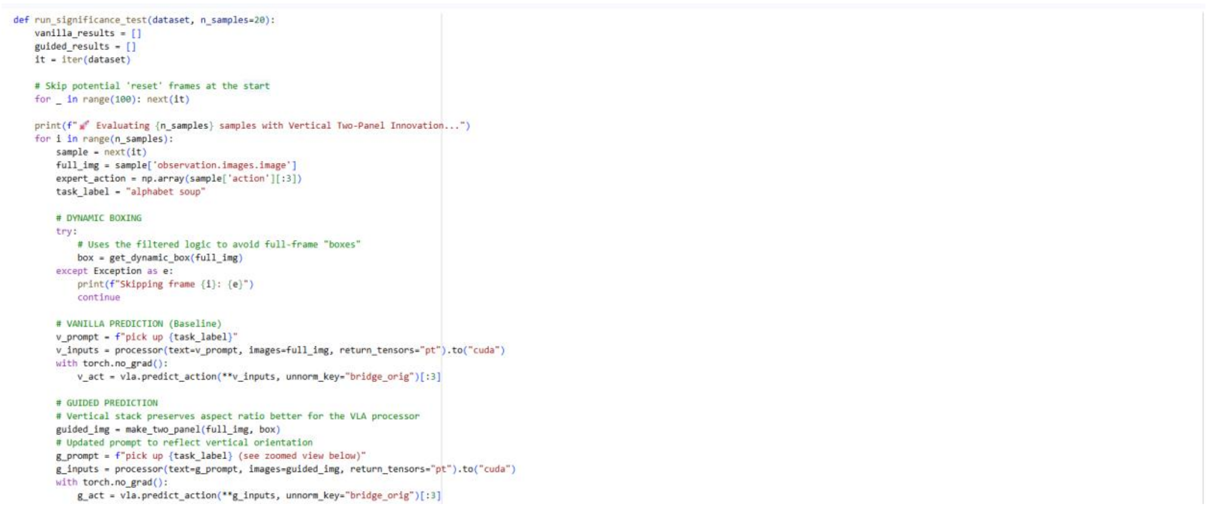


```python
it = iter(ds)
sample = next(it)
img = sample['observation.images.image']
box = get_dynamic_box(img)
guided = make_two_panel(img, box)

print(f"Detected Box: {box}")
plt.figure(figsize=(10, 5))
plt.subplot(1, 2, 1); plt.imshow(img); plt.title("Input")
plt.subplot(1, 2, 2); plt.imshow(guided); plt.title("Guided (Zoomed)")
plt.show()
```

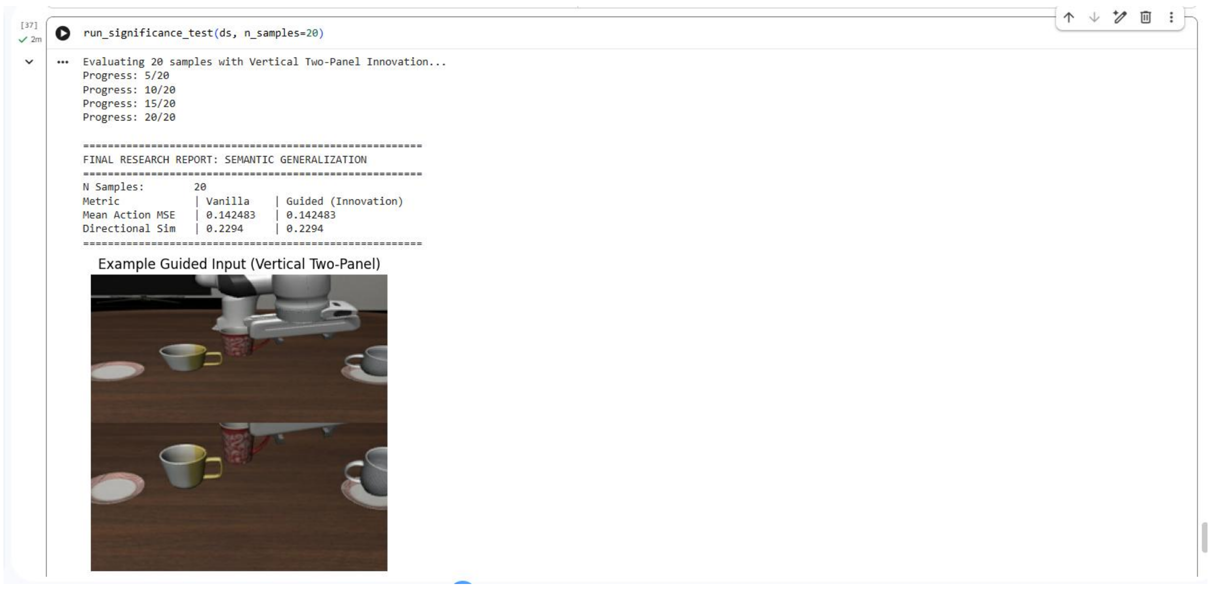
```
Detected Box: [np.int64(0), np.int64(112), np.int64(255), np.int64(255)]
```



This code evaluates the "Vertical Two-Panel" innovation on the OpenVLA model's performance. It checks whether providing both global and local (zoomed) visual contexts, processed via a vertical stack to preserve aspect ratio improves robotic movement accuracy.



```python
def run_significance_test(dataset, n_samples=20):
    vanilla_results = []
    guided_results = []
    it = iter(dataset)

    # Skip potential 'reset' frames at the start
    for _ in range(100): next(it)

    print(f"🚀 Evaluating {n_samples} samples with Vertical Two-Panel Innovation...")
    for i in range(n_samples):
        sample = next(it)
        full_img = sample['observation.images.image']
        expert_action = np.array(sample['action'][:3])
        task_label = "alphabet soup"

        # DYNAMIC BOXING
        try:
            # Uses the filtered logic to avoid full-frame "boxes"
            box = get_dynamic_box(full_img)
        except Exception as e:
            print(f"Skipping frame {i}: {e}")
            continue

        # VANILLA PREDICTION (Baseline)
        v_prompt = f"pick up {task_label}"
        v_inputs = processor(text=v_prompt, images=full_img, return_tensors="pt").to("cuda")
        with torch.no_grad():
            v_act = vla.predict_action(**v_inputs, unnorm_key="bridge_orig")[:3]

        # GUIDED PREDICTION
        # Vertical stack preserves aspect ratio better for the VLA processor
        guided_img = make_two_panel(full_img, box)
        # Updated prompt to reflect vertical orientation
        g_prompt = f"pick up {task_label} (see zoomed view below)"
        g_inputs = processor(text=g_prompt, images=guided_img, return_tensors="pt").to("cuda")
        with torch.no_grad():
            g_act = vla.predict_action(**g_inputs, unnorm_key="bridge_orig")[:3]
```



```
[37]
run_significance_test(ds, n_samples=20)
```

```
Evaluating 20 samples with Vertical Two-Panel Innovation...
Progress: 5/20
Progress: 10/20
Progress: 15/20
Progress: 20/20

=======================================================
FINAL RESEARCH REPORT: SEMANTIC GENERALIZATION
=======================================================
N Samples:        20
Metric            | Vanilla    | Guided (Innovation)
Mean Action MSE   | 0.142483   | 0.142483
Directional Sim   | 0.2294     | 0.2294
=======================================================
```

Example Guided Input (Vertical Two-Panel)

Hemaank Mahajan

28.1.2026