UGround vs Qwen2-VL Evaluation on Click-100k

This notebook compares the 7B versions of:

- **osunlp/UGround-V1-7B** (update the model id below if your 7B checkpoint has a different name)
- **Qwen/Qwen2-VL-7B-Instruct**

Runs are sequential (one vLLM server at a time) because two 7B models cannot fit on the same GPU concurrently. The dataset remains the first 100 samples from `mlfoundations/Click-100k`.

In [1]:
```
import re
import numpy as np
from PIL import Image
from datasets import load_dataset
from openai import OpenAI
from tqdm import tqdm
import matplotlib.pyplot as plt
import pandas as pd
from typing import Tuple, Dict, List
import base64
from io import BytesIO
```

In [2]:
```
# Model configurations for vLLM OpenAI endpoints (7B, sequential)
MODEL_CONFIGS = {
    "uground-7b": {
        "model_id": "osunlp/UGround-V1-7B",  # update if your 7B checkpoint name differs
        "endpoint": "http://localhost:8000/v1",  # reused sequentially
    },
    "qwen2-vl-7b": {
        "model_id": "Qwen/Qwen2-VL-7B-Instruct",
        "endpoint": "http://localhost:8000/v1",  # reused sequentially
    },
    "ui-tars-7b": {
        "model_id": "ByteDance-Seed/UI-TARS-7B-DPO",
        "endpoint": "http://localhost:8001/v1",  # reused sequentially
    }
}
```

```python
# Order to run models (one at a time)
RUN_ORDER = ["uground-7b", "qwen2-vl-7b", "ui-tars-7b"]

print("Using vLLM OpenAI-compatible hosting (one server at a time; sequential 7B runs)")
print("Update MODEL_CONFIGS if your checkpoint names or ports differ.")

# Helper utilities to connect/verify a single model server when it is running

def verify_vllm_server_connection(client, model_key, expected_model_id):
    models = client.models.list()
    ids = [m.id for m in getattr(models, "data", [])]
    if not ids:
        raise RuntimeError(f"Connection to {model_key} succeeded but returned no models.")
    print(f"✓ {model_key} connected: available models {ids}")
    if expected_model_id not in ids:
        print(f"⚠ Expected {expected_model_id} but server reported {ids}")


def connect_client(model_key: str) -> OpenAI:
    cfg = MODEL_CONFIGS[model_key]
    endpoint = cfg["endpoint"]
    print(f"Connecting to {model_key} @ {endpoint} (expected model: {cfg['model_id']})")
    client = OpenAI(base_url=endpoint, api_key="EMPTY")
    verify_vllm_server_connection(client, model_key, cfg["model_id"])
    return client

# Clients will be created inside the evaluation loop to keep runs sequential
clients = {}
```

```
Using vLLM OpenAI-compatible hosting (one server at a time; sequential 7B runs)
Update MODEL_CONFIGS if your checkpoint names or ports differ.
```

2. Helper Functions

In [3]:

```python
def image_to_base64(image: Image.Image) -> str:
    """Convert PIL Image to base64 data URL."""
    buffered = BytesIO()
    image.save(buffered, format="PNG")
    img_str = base64.b64encode(buffered.getvalue()).decode()
    return f"{img_str}"
```

```python
def build_uground_messages(description: str, image_url: str):
    """
    Build chat messages in OpenAI format for vision-language models.
    """
    return [
        {
            "role": "user",
            "content": [
                {
                    "type": "image_url",
                    "image_url": {"url": image_url}
                },
                {
                    "type": "text",
                    "text": f"""Find the coordinates of the element described below. Return ONLY the coordi

Description: {description}

Coordinates:"""
                },
            ],
        },
    ]


def call_model_raw(
    image: Image.Image,
    query: str,
    client: OpenAI,
    model_name: str,
) -> Tuple[str, int, int]:
    """
    Send one screenshot + grounding query to SGLang API, return:
    (raw_text_response, orig_width, orig_height).
    """
    orig_w, orig_h = image.size

    # Convert image to base64
    image_url = image_to_base64(image)
```

```python
    # Build messages in OpenAI format
    messages = build_uground_messages(query, image_url)

    try:
        # Call SGLang API (OpenAI-compatible)
        response = client.chat.completions.create(
            model=model_name,
            messages=messages,
            max_tokens=128,
            temperature=0.0,
        )

        reply = response.choices[0].message.content.strip()
    except Exception as e:
        print(f"API call error: {e}")
        reply = ""

    return reply, orig_w, orig_h


def parse_xy_from_string(text: str) -> Tuple[int, int]:
    """
    Extract (x, y) coordinates from a string.
    Handles various formats:
    - (x, y) - two coordinates
    - (x1, y1, x2, y2) - four coordinates (bbox), computes center
    - Multiple coordinate pairs - takes the first valid one
    """
    if not text or not text.strip():
        raise ValueError(f"Empty text provided")

    # Clean the text - remove extra whitespace and newlines
    text = text.strip()

    # Try to find coordinate patterns: (num, num) or (num, num, num, num)
    # Pattern 1: (x, y) - two coordinates
    pattern_two = re.findall(r"\(\s*(\d+)\s*,\s*(\d+)\s*\)", text)
    # Pattern 2: (x1, y1, x2, y2) - four coordinates (bbox)
    pattern_four = re.findall(r"\(\s*(\d+)\s*,\s*(\d+)\s*,\s*(\d+)\s*,\s*(\d+)\s*\)", text)

    # First try to find a simple (x, y) pair in [0, 1000) range
    if pattern_two:
```

```python
        for match in pattern_two:
            x, y = int(match[0]), int(match[1])
            if 0 <= x < 1000 and 0 <= y < 1000:
                return x, y

    # If we found a bbox format (x1, y1, x2, y2), compute center
    if pattern_four:
        for match in pattern_four:
            x1, y1, x2, y2 = int(match[0]), int(match[1]), int(match[2]), int(match[3])
            # Compute center
            center_x = (x1 + x2) // 2
            center_y = (y1 + y2) // 2
            # If coordinates are in [0, 1000) range, use center
            if 0 <= center_x < 1000 and 0 <= center_y < 1000:
                return center_x, center_y
            # If coordinates are in pixel space (> 1000), use first two if in range
            elif 0 <= x1 < 1000 and 0 <= y1 < 1000:
                return x1, y1

    # Fallback: try to extract just the first two numbers
    numbers = re.findall(r'\d+', text)
    if len(numbers) >= 2:
        x, y = int(numbers[0]), int(numbers[1])
        # Clamp to [0, 1000) range
        x = max(0, min(999, x))
        y = max(0, min(999, y))
        return x, y

    raise ValueError(f"Could not parse valid coordinates from: {text[:200]!r}")


def scale_to_pixels(x_1000: int, y_1000: int, width: int, height: int) -> Tuple[int, int]:
    """
    Map model coordinates in [0,1000) to pixel coordinates of the original image.
    """
    x_px = int(x_1000 / 1000 * width)
    y_px = int(y_1000 / 1000 * height)
    x_px = max(0, min(width - 1, x_px))
    y_px = max(0, min(height - 1, y_px))
    return x_px, y_px
```

```python
def predict_coordinates(
    image: Image.Image,
    query: str,
    model_key: str,
) -> Tuple[int, int, str]:
    """
    Predict coordinates using the specified model via vLLM OpenAI endpoint.
    Returns: (x_px, y_px, raw_response)
    """
    cfg = MODEL_CONFIGS[model_key]
    client = clients[model_key]
    model_name = cfg["model_id"]

    reply, orig_w, orig_h = call_model_raw(image, query, client, model_name)

    # If reply is empty, return center as fallback
    if not reply or reply.strip() == "":
        return orig_w // 2, orig_h // 2, reply

    try:
        x_1000, y_1000 = parse_xy_from_string(reply)
        x_px, y_px = scale_to_pixels(x_1000, y_1000, orig_w, orig_h)
        return x_px, y_px, reply
    except ValueError as e:
        # If parsing fails, return center of image as fallback
        # Only print warning if reply is not empty (to avoid spam)
        if reply.strip():
            print(f"Warning: Failed to parse coordinates from {model_key}: {e}, reply: {reply[:50]}")
        return orig_w // 2, orig_h // 2, reply


def euclidean_distance(x1: int, y1: int, x2: int, y2: int) -> float:
    """Calculate Euclidean distance between two points."""
    return np.sqrt((x1 - x2) ** 2 + (y1 - y2) ** 2)


def bbox_to_center(bbox: List[int]) -> Tuple[int, int]:
    """
    Convert bounding box [x1, y1, x2, y2] to center coordinates (x, y).
    """
    if len(bbox) != 4:
        raise ValueError(f"Bbox must have 4 elements, got {len(bbox)}")
```

```
    x1, y1, x2, y2 = bbox
    center_x = (x1 + x2) // 2
    center_y = (y1 + y2) // 2
    return center_x, center_y
```

3. Load Click-100k Dataset

In [4]:
```
print("Loading Click-100k dataset...")
dataset = load_dataset("mlfoundations/Click-100k", split="train")
print(f"Total samples in dataset: {len(dataset)}")

# Take first 100 samples
eval_dataset = dataset.select(range(min(100, len(dataset))))
print(f"Evaluating on {len(eval_dataset)} samples")
print(f"\nDataset features: {eval_dataset.features}")
print(f"\nSample keys: {eval_dataset[0].keys()}")
```

```
Loading Click-100k dataset...
Resolving data files:   0%|          | 0/64 [00:00<?, ?it/s]
Resolving data files:   0%|          | 0/64 [00:00<?, ?it/s]
Loading dataset shards:   0%|          | 0/58 [00:00<?, ?it/s]
Total samples in dataset: 101314
Evaluating on 100 samples

Dataset features: {'image_path': Value('string'), 'images': List(Image(mode=None, decode=True)), 'easyr1_pr
ompt': Value('string'), 'bbox': List(Value('int64')), 'image_width': Value('int64'), 'image_height': Value
('int64'), 'normalized_bbox': List(Value('float64'))}

Sample keys: dict_keys(['image_path', 'images', 'easyr1_prompt', 'bbox', 'image_width', 'image_height', 'no
rmalized_bbox'])
```

4. Run Evaluation

Run one vLLM server at a time (sequential 7B runs):

```
cd /workspace/CSE291A_ECUA3
source /root/miniconda3/etc/profile.d/conda.sh
conda activate ecua
```

```
# Step A: UGround 7B (run this first)
# Update the model id below if your 7B checkpoint name differs
python -m vllm.entrypoints.openai.api_server \
  --model osunlp/UGround-V1-7B \
  --trust-remote-code \
  --port 8000 \
  --host 0.0.0.0 \
  --gpu-memory-utilization 0.9 \
  --max-num-seqs 1

# After UGround finishes, stop it:
pkill -f "vllm.entrypoints.openai.api_server"

# Step B: Qwen2-VL 7B (run after UGround is done)
python -m vllm.entrypoints.openai.api_server \
  --model Qwen/Qwen2-VL-7B-Instruct \
  --trust-remote-code \
  --port 8000 \
  --host 0.0.0.0 \
  --gpu-memory-utilization 0.9 \
  --max-num-seqs 1

# After Qwen finishes, stop it:
pkill -f "vllm.entrypoints.openai.api_server"

# Step C: UI-TARS 7B (run after Qwen is done)
python -m vllm.entrypoints.openai.api_server \
  --model ByteDance-Seed/UI-TARS-7B-DPO \
  --trust-remote-code \
  --port 8000 \
  --host 0.0.0.0 \
  --gpu-memory-utilization 0.9 \
  --max-num-seqs 1
```

Notes:

- Only one 7B model should be running at any time. We reuse port `8000` for both.
- The notebook will prompt you (via `input`) before each model run so you can start/stop the matching vLLM server.
- Uses cached weights from `/workspace/cache/huggingface` if available.

- If your 7B model id differs (e.g., a fine-tuned checkpoint), update the `MODEL_CONFIGS` cell below.

In [6]:
```python
# Initialize results storage (sequential 7B runs)
results = {key: [] for key in RUN_ORDER}

print("Running evaluation sequentially (one model at a time)...")
for model_key in RUN_ORDER:
    cfg = MODEL_CONFIGS[model_key]
    input(f"\nStart vLLM server for {cfg['model_id']} at {cfg['endpoint']} then press Enter...")
    clients[model_key] = connect_client(model_key)

    print(f"\nProcessing samples with {model_key}...")
    for idx in tqdm(range(len(eval_dataset)), desc=f"Processing samples ({model_key})"):
        sample = eval_dataset[idx]

        # Extract image from dataset - images is a list, take the first one
        images_list = sample.get("images")
        if images_list is None or len(images_list) == 0:
            print(f"Skipping sample {idx}: no images found")
            continue

        # Get the first image from the list
        image = images_list[0]
        if not isinstance(image, Image.Image):
            if hasattr(image, "convert"):
                image = image.convert("RGB")
            else:
                print(f"Skipping sample {idx}: cannot process image format")
                continue

        # Extract query
        query = sample.get("easyr1_prompt")
        if query is None:
            print(f"Skipping sample {idx}: missing query")
            continue

        # Extract bbox and convert to center coordinates
        bbox = sample.get("bbox")
        if bbox is None or len(bbox) != 4:
            print(f"Skipping sample {idx}: invalid bbox {bbox}")
            continue
```

```
        try:
            gt_x, gt_y = bbox_to_center(bbox)
        except Exception as e:
            print(f"Skipping sample {idx}: error converting bbox: {e}")
            continue

        try:
            pred_x, pred_y, raw_response = predict_coordinates(image, query, model_key)
            distance = euclidean_distance(pred_x, pred_y, gt_x, gt_y)

            results[model_key].append({
                "sample_idx": idx,
                "query": query,
                "bbox": bbox,
                "gt_x": gt_x,
                "gt_y": gt_y,
                "pred_x": pred_x,
                "pred_y": pred_y,
                "distance": distance,
                "raw_response": raw_response,
            })
        except Exception as e:
            print(f"Error processing sample {idx} with {model_key}: {e}")
            results[model_key].append({
                "sample_idx": idx,
                "query": query,
                "bbox": bbox,
                "gt_x": gt_x,
                "gt_y": gt_y,
                "pred_x": None,
                "pred_y": None,
                "distance": float('inf'),
                "raw_response": str(e),
                "error": True,
            })

    print(f"Completed {model_key}.")

print("\nSequential evaluation complete!")
```

```
Running evaluation sequentially (one model at a time)...
Connecting to uground-7b @ http://localhost:8000/v1 (expected model: osunlp/UGround-V1-7B)
✓ uground-7b connected: available models ['osunlp/UGround-V1-7B']

Processing samples with uground-7b...
Processing samples (uground-7b): 100%|██████████| 100/100 [02:14<00:00,  1.34s/it]
Completed uground-7b.
Connecting to qwen2-vl-7b @ http://localhost:8000/v1 (expected model: Qwen/Qwen2-VL-7B-Instruct)
✓ qwen2-vl-7b connected: available models ['Qwen/Qwen2-VL-7B-Instruct']

Processing samples with qwen2-vl-7b...
Processing samples (qwen2-vl-7b): 100%|██████████| 100/100 [02:25<00:00,  1.46s/it]
Completed qwen2-vl-7b.
Connecting to ui-tars-7b @ http://localhost:8001/v1 (expected model: ByteDance-Seed/UI-TARS-7B-DPO)
✓ ui-tars-7b connected: available models ['ByteDance-Seed/UI-TARS-7B-DPO']

Processing samples with ui-tars-7b...
Processing samples (ui-tars-7b): 100%|██████████| 100/100 [02:06<00:00,  1.26s/it]
Completed ui-tars-7b.

Sequential evaluation complete!
```

5. Calculate Metrics

In [7]:

```python
def calculate_metrics(results: List[Dict]) -> Dict:
    """Calculate evaluation metrics."""
    distances = [r["distance"] for r in results if r.get("distance") != float('inf')]

    if not distances:
        return {
            "mean_distance": float('inf'),
            "median_distance": float('inf'),
            "std_distance": float('inf'),
            "accuracy_5px": 0.0,
            "accuracy_10px": 0.0,
            "accuracy_20px": 0.0,
            "accuracy_50px": 0.0,
            "total_samples": len(results),
            "successful_samples": 0,
```

```
        }

    distances = np.array(distances)

    # Calculate accuracy at different thresholds
    thresholds = [5, 10, 20, 50]
    accuracies = {}
    for threshold in thresholds:
        accuracies[f"accuracy_{threshold}px"] = np.mean(distances <= threshold) * 100

    return {
        "mean_distance": float(np.mean(distances)),
        "median_distance": float(np.median(distances)),
        "std_distance": float(np.std(distances)),
        **accuracies,
        "total_samples": len(results),
        "successful_samples": len(distances),
    }

# Calculate metrics for all available models in results
metrics = {}
for model_key in MODEL_CONFIGS.keys():
    metrics[model_key] = calculate_metrics(results[model_key])
    print(f"\n{model_key.upper()} Metrics:")
    print(f"  Mean Distance Error: {metrics[model_key]['mean_distance']:.2f} px")
    print(f"  Median Distance Error: {metrics[model_key]['median_distance']:.2f} px")
    print(f"  Std Distance Error: {metrics[model_key]['std_distance']:.2f} px")
    print(f"  Accuracy @ 5px: {metrics[model_key]['accuracy_5px']:.2f}%")
    print(f"  Accuracy @ 10px: {metrics[model_key]['accuracy_10px']:.2f}%")
    print(f"  Accuracy @ 20px: {metrics[model_key]['accuracy_20px']:.2f}%")
    print(f"  Accuracy @ 50px: {metrics[model_key]['accuracy_50px']:.2f}%")
    print(f"  Successful Predictions: {metrics[model_key]['successful_samples']}/{metrics[model_key]['tota
```

```
UGROUND-7B Metrics:
  Mean Distance Error: 266.46 px
  Median Distance Error: 61.98 px
  Std Distance Error: 417.94 px
  Accuracy @ 5px: 30.00%
  Accuracy @ 10px: 35.00%
  Accuracy @ 20px: 38.00%
  Accuracy @ 50px: 48.00%
  Successful Predictions: 100/100

QWEN2-VL-7B Metrics:
  Mean Distance Error: 312.34 px
  Median Distance Error: 161.85 px
  Std Distance Error: 370.60 px
  Accuracy @ 5px: 0.00%
  Accuracy @ 10px: 2.00%
  Accuracy @ 20px: 4.00%
  Accuracy @ 50px: 22.00%
  Successful Predictions: 100/100

UI-TARS-7B Metrics:
  Mean Distance Error: 152.11 px
  Median Distance Error: 26.62 px
  Std Distance Error: 268.25 px
  Accuracy @ 5px: 16.00%
  Accuracy @ 10px: 30.00%
  Accuracy @ 20px: 45.00%
  Accuracy @ 50px: 60.00%
  Successful Predictions: 100/100
```

6. Comparison Table

In [9]:

```python
# Create comparison DataFrame
comparison_data = []
for model_key in MODEL_CONFIGS.keys():
    m = metrics[model_key]
    comparison_data.append({
        "Model": model_key,
        "Mean Distance (px)": f"{m['mean_distance']:.2f}",
        "Median Distance (px)": f"{m['median_distance']:.2f}",
```

```
        "Std Distance (px)": f"{m['std_distance']:.2f}",
        "Accuracy @ 5px (%)": f"{m['accuracy_5px']:.2f}",
        "Accuracy @ 10px (%)": f"{m['accuracy_10px']:.2f}",
        "Accuracy @ 20px (%)": f"{m['accuracy_20px']:.2f}",
        "Accuracy @ 50px (%)": f"{m['accuracy_50px']:.2f}",
        "Success Rate": f"{m['successful_samples']}/{m['total_samples']}",
    })

df_comparison = pd.DataFrame(comparison_data)
print("\n" + "="*80)
print("MODEL COMPARISON")
print("="*80)
print(df_comparison.to_string(index=False))
print("="*80)
```

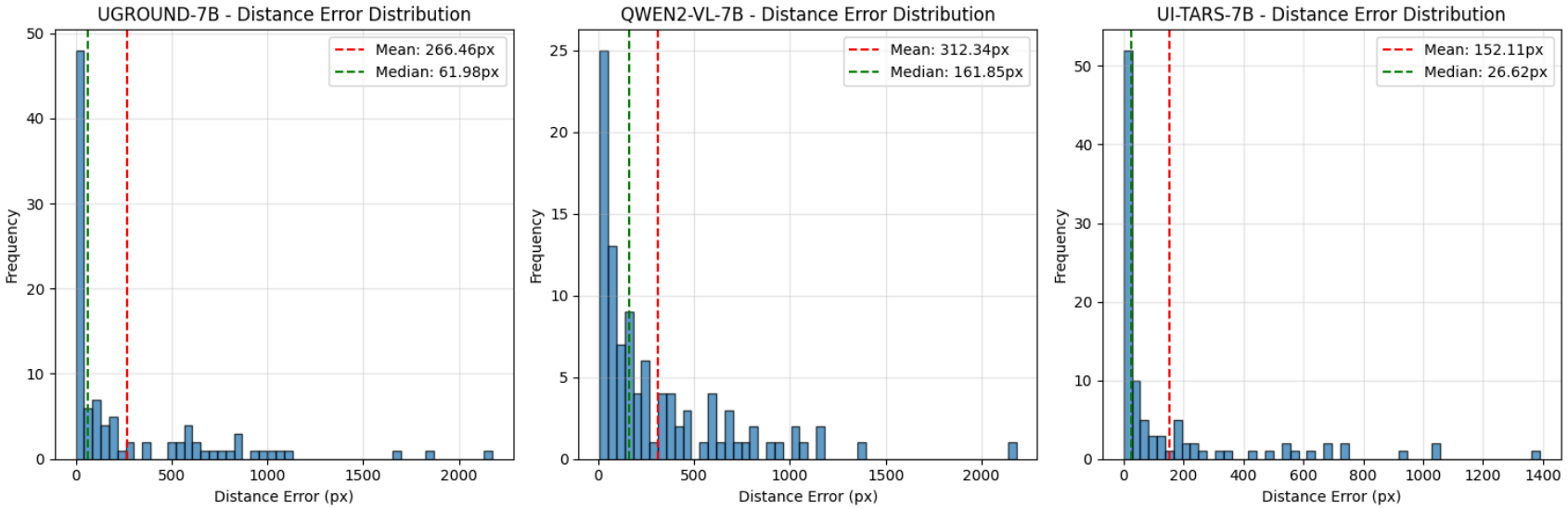
```
================================================================================
MODEL COMPARISON
================================================================================
      Model Mean Distance (px) Median Distance (px) Std Distance (px) Accuracy @ 5px (%) Accuracy @ 10px
(%) Accuracy @ 20px (%) Accuracy @ 50px (%) Success Rate
 uground-7b             266.46                61.98            417.94              30.00               35.0
0               38.00               48.00      100/100
qwen2-vl-7b             312.34               161.85            370.60               0.00                2.0
0                4.00               22.00      100/100
 ui-tars-7b             152.11                26.62            268.25              16.00               30.0
0               45.00               60.00      100/100
================================================================================
```

7. Visualization

In [11]:

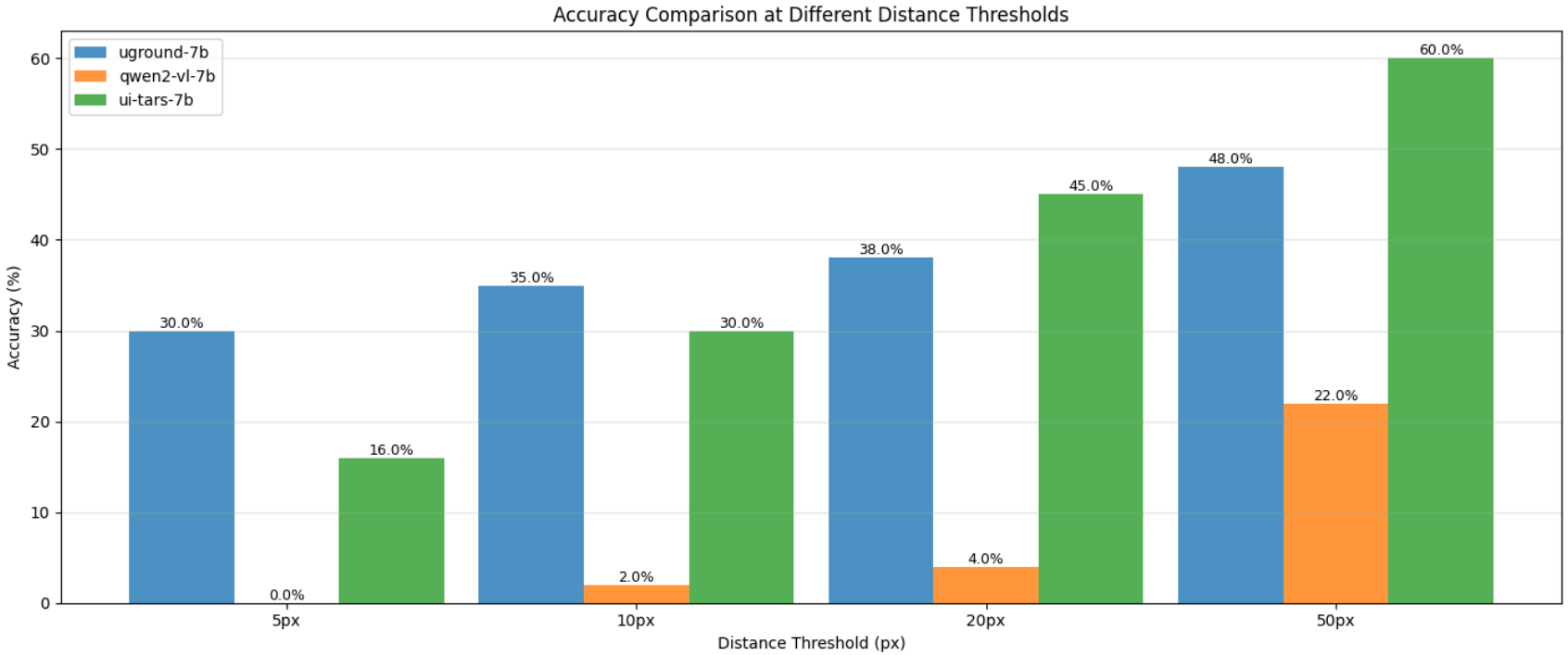
```
# Plot distance distributions
fig, axes = plt.subplots(1, 3, figsize=(15, 5))

for idx, model_key in enumerate(MODEL_CONFIGS.keys()):
    distances = [r["distance"] for r in results[model_key]
                 if r.get("distance") != float('inf')]

    if distances:
        axes[idx].hist(distances, bins=50, alpha=0.7, edgecolor='black')
        axes[idx].axvline(np.mean(distances), color='red', linestyle='--',
                          label=f'Mean: {np.mean(distances):.2f}px')
```

```
        axes[idx].axvline(np.median(distances), color='green', linestyle='--',
                          label=f'Median: {np.median(distances):.2f}px')
        axes[idx].set_xlabel('Distance Error (px)')
        axes[idx].set_ylabel('Frequency')
        axes[idx].set_title(f'{model_key.upper()} - Distance Error Distribution')
        axes[idx].legend()
        axes[idx].grid(True, alpha=0.3)
    else:
        axes[idx].text(0.5, 0.5, 'No valid predictions',
                       ha='center', va='center', transform=axes[idx].transAxes)
        axes[idx].set_title(f'{model_key.upper()} - No Data')

plt.tight_layout()
plt.show()
```



In [13]:
```
# Plot accuracy comparison
model_keys = [k for k in metrics.keys()] or list(metrics.keys())
fig, ax = plt.subplots(figsize=(8 + 2 * len(model_keys), 6))

thresholds = [5, 10, 20, 50]
x = np.arange(len(thresholds))
width = 0.3

bars_all = []
for idx, model_key in enumerate(model_keys):
```

```
    offsets = (idx - (len(model_keys) - 1) / 2) * width
    accs = [metrics[model_key][f"accuracy_{t}px"] for t in thresholds]
    bars = ax.bar(x + offsets, accs, width, label=model_key, alpha=0.8)
    bars_all.append(bars)

ax.set_xlabel('Distance Threshold (px)')
ax.set_ylabel('Accuracy (%)')
ax.set_title('Accuracy Comparison at Different Distance Thresholds')
ax.set_xticks(x)
ax.set_xticklabels([f'{t}px' for t in thresholds])
ax.legend()
ax.grid(True, alpha=0.3, axis='y')

# Add value labels on bars
for bars in bars_all:
    for bar in bars:
        height = bar.get_height()
        ax.text(bar.get_x() + bar.get_width()/2., height,
                f'{height:.1f}%',
                ha='center', va='bottom', fontsize=9)

plt.tight_layout()
plt.show()
```



8. Sample Predictions

In [14]:

```python
# Show some example predictions
num_examples = min(5, len(eval_dataset))
print(f"\nShowing {num_examples} example predictions:\n")

for idx in range(num_examples):
    sample = eval_dataset[idx]
    images_list = sample.get("images")
    query = sample.get("easyr1_prompt")
    bbox = sample.get("bbox")

    if images_list is None or len(images_list) == 0 or query is None or bbox is None:
        continue

    image = images_list[0]
    if not isinstance(image, Image.Image):
        if hasattr(image, "convert"):
```

```
            image = image.convert("RGB")
        else:
            continue

    try:
        gt_x, gt_y = bbox_to_center(bbox)
    except:
        continue

    print(f"\n{'='*80}")
    print(f"Example {idx + 1}")
    print(f"{'='*80}")
    print(f"Query: {query}")
    print(f"Bbox: {bbox}")
    print(f"Ground Truth Center: ({gt_x}, {gt_y})")

    for model_key in MODEL_CONFIGS.keys():
        # Find the result for this sample index
        result = None
        for r in results[model_key]:
            if r.get("sample_idx") == idx:
                result = r
                break

        if result is None:
            print(f"\n{model_key}: No result found")
            continue

        if result.get("error"):
            print(f"\n{model_key}: ERROR - {result['raw_response']}")
        else:
            print(f"\n{model_key}:")
            print(f"  Ground Truth: ({result['gt_x']}, {result['gt_y']})")
            print(f"  Predicted: ({result['pred_x']}, {result['pred_y']})")
            print(f"  Distance Error: {result['distance']:.2f} px")
            print(f"  Raw Response: {result['raw_response']}")
```

Showing 5 example predictions:

==
Example 1
==
Query: You are an expert UI element locator. Given a GUI image and a user's element description, provide th
e coordinates of the specified element as a single (x,y) point. For elements with area, return the center p
oint.

Output the coordinate pair exactly:
(x,y)

<image>
Tap on Located between 'Background' and 'Notifications' options.
Bbox: [72, 183, 322, 232]
Ground Truth Center: (197, 207)

uground-7b:
  Ground Truth: (197, 207)
  Predicted: (196, 210)
  Distance Error: 3.16 px
  Raw Response: (100, 193)

qwen2-vl-7b:
  Ground Truth: (197, 207)
  Predicted: (80, 131)
  Distance Error: 139.52 px
  Raw Response: (41, 120)

ui-tars-7b:
  Ground Truth: (197, 207)
  Predicted: (203, 209)
  Distance Error: 6.32 px
  Raw Response: (104,192)

==
Example 2
==
Query: You are an expert UI element locator. Given a GUI image and a user's element description, provide th
e coordinates of the specified element as a single (x,y) point. For elements with area, return the center p
oint.

Output the coordinate pair exactly:
(x,y)

<image>
I will give textual descriptions of a certain element in the screenshot. Please predict the location of the corresponding element (with point). This element initiates a search function on the webpage, allowing users to search for specific content or topics.
Bbox: [432, 2, 449, 20]
Ground Truth Center: (440, 11)

uground-7b:
  Ground Truth: (440, 11)
  Predicted: (440, 11)
  Distance Error: 0.00 px
  Raw Response: (926, 41)

qwen2-vl-7b:
  Ground Truth: (440, 11)
  Predicted: (264, 2)
  Distance Error: 176.23 px
  Raw Response: (555, 10)

ui-tars-7b:
  Ground Truth: (440, 11)
  Predicted: (446, 15)
  Distance Error: 7.21 px
  Raw Response: (937,54)

==
Example 3
==
Query: You are an expert UI element locator. Given a GUI image and a user's element description, provide th
e coordinates of the specified element as a single (x,y) point. For elements with area, return the center p
oint.

Output the coordinate pair exactly:
(x,y)

<image>
Given the following text: "Is Confucian democracy possible?" , find the text in the document and click the
1th character "u" in the text.

```
Bbox: [769, 998, 779, 1009]
Ground Truth Center: (774, 1003)

uground-7b:
  Ground Truth: (774, 1003)
  Predicted: (842, 994)
  Distance Error: 68.59 px
  Raw Response: (436, 911)

qwen2-vl-7b:
  Ground Truth: (774, 1003)
  Predicted: (852, 1002)
  Distance Error: 78.01 px
  Raw Response: (441,918)

ui-tars-7b:
  Ground Truth: (774, 1003)
  Predicted: (767, 994)
  Distance Error: 11.40 px
  Raw Response: (397,911)

================================================================================
Example 4
================================================================================
Query: You are an expert UI element locator. Given a GUI image and a user's element description, provide th
e coordinates of the specified element as a single (x,y) point. For elements with area, return the center p
oint.

Output the coordinate pair exactly:
(x,y)

<image>
Locate Health Sleep
Bbox: [550, 68, 571, 78]
Ground Truth Center: (560, 73)

uground-7b:
  Ground Truth: (560, 73)
  Predicted: (558, 72)
  Distance Error: 2.24 px
  Raw Response: (554, 129)
```

```
qwen2-vl-7b:
  Ground Truth: (560, 73)
  Predicted: (554, 56)
  Distance Error: 18.03 px
  Raw Response: (550, 100)

ui-tars-7b:
  Ground Truth: (560, 73)
  Predicted: (582, 73)
  Distance Error: 22.00 px
  Raw Response: (578,131)

================================================================================
Example 5
================================================================================
Query: You are an expert UI element locator. Given a GUI image and a user's element description, provide th
e coordinates of the specified element as a single (x,y) point. For elements with area, return the center p
oint.

Output the coordinate pair exactly:
(x,y)

<image>
With this screenshot of a webpage, can you locate the element I describe (with point)? This element provide
s access to detailed information about the platform's privacy practices.
Bbox: [145, 340, 203, 355]
Ground Truth Center: (174, 347)

uground-7b:
  Ground Truth: (174, 347)
  Predicted: (172, 347)
  Distance Error: 2.00 px
  Raw Response: (193, 690)

qwen2-vl-7b:
  Ground Truth: (174, 347)
  Predicted: (403, 5)
  Distance Error: 411.59 px
  Raw Response: (450, 10)

ui-tars-7b:
  Ground Truth: (174, 347)
```

Predicted: (185, 347)
Distance Error: 11.00 px
Raw Response: (207,690)