

Caltech



Cornell Bowers CIS
Computer Science



TEXAS
The University of Texas at Austin

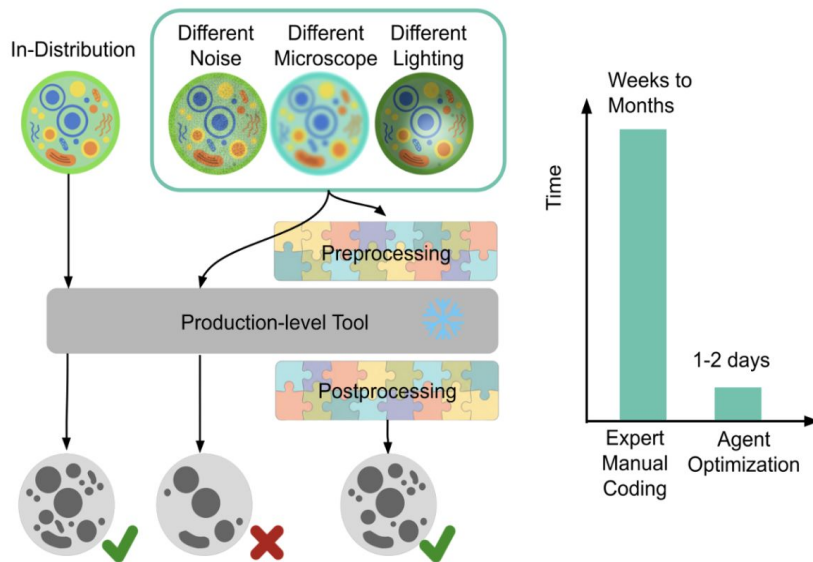


Simple Agents Outperform *Experts* in Biomedical Imaging Workflow Optimization

Xuefei (Julie) Wang¹, Kai A. Horstmann², Ethan Lin², Jonathan Chen², Alexander R. Farhang¹, Sophia Stiles¹, Atharva Sehgal³, Jonathan Light⁴, David Van Valen¹, Yisong Yue¹, Jennifer J. Sun²

Caltech¹, Cornell², UT Austin³, Rensselaer Polytechnic Institute⁴

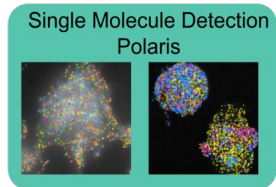
Motivation



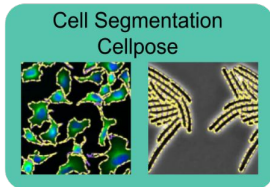
- Adapting production-level tools to new, *out-of-distribution* scientific datasets is a critical **“last-mile” bottleneck**.
- We consider using **AI agents** to automate this task.

Framework

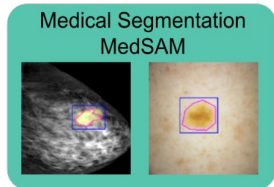
Tasks



Scale
Molecular
Cellular
Macroscopic

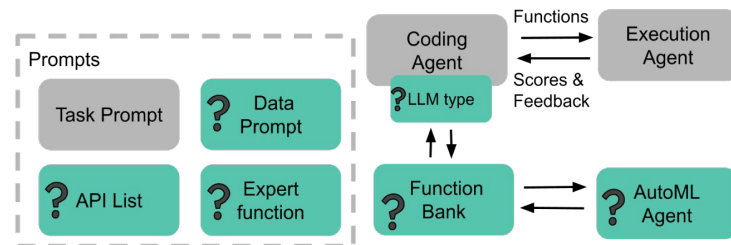


Task
Single molecule detection
Cell segmentation
Medical segmentation



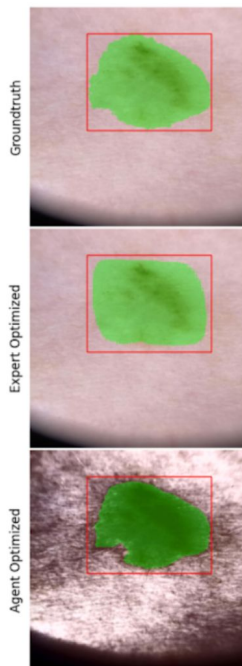
Tool
Polaris
Cellpose
MedSAM

Agent Design Space



Results

	Expert Baseline	Base Agent
Polaris (F1)	0.841	0.867
Cellpose (AP@IoU 0.5)	0.402	0.409
MedSAM (NSD+DSC)	0.820	0.971



Expert optimized - preprocessing

```

1 import numpy as np
2 def preprocess_images(images, is_rgb=False):
3     resized_imgs = images
4     for i in range(len(resized_imgs)):
5         img_np = resized_imgs[i]
6         if is_rgb:
7             resized_imgs[i] = np.uint8((img_np - img_np.min()) / (np.max(img_np)
8             - np.min(img_np)) * 255.0)
9         else:
10            lower_bound, upper_bound = np.percentile(img_np[img_np > 0], 0.5), np
11            .percentile(img_np[img_np > 0], 99.5)
12            img_np_pre = np.clip(img_np, lower_bound, upper_bound)
13            img_np_pre = (img_np_pre - np.min(img_np_pre)) / (np.max(img_np_pre
14            ) - np.min(img_np_pre)) * 255.0
15            img_np_pre[img_np == 0] = 0
16            resized_imgs[i] = np.uint8(img_np_pre)
17     return resized_imgs

```

Percentile-based Normalization

Expert optimized - postprocessing

```

1 import torch.nn.functional as F
2 def postprocess_preds(preds):
3     upsampled = F.interpolate(preds, size=(512, 512), mode='bilinear',
4     align_corners=False)
5     upsampled = upsampled.squeeze(1)
6     seg = (upsampled > 0.5).int()
7     return seg

```

Bilinear Interpolation

Agent optimized - preprocessing

```

1 import numpy as np
2 import cv2
3 def preprocess_images(images):
4     processed_images = []
5     for img_array in images:
6         img_array = np.copy(img_array)
7         min_val = img_array.min()
8         ptp_val = np.ptp(img_array)
9         img_norm = (img_array - min_val) / (ptp_val + 1e-8)
10        img_8bit = (img_norm * 255).astype(np.uint8)
11        img_ycrCb = cv2.cvtColor(img_8bit, cv2.COLOR_RGB2YCrCb)
12        y, Cr, Cb = cv2.split(img_ycrCb)
13        y = cv2.equalizeHist(y)
14        img_ycrCb_eq = cv2.merge([y, Cr, Cb])
15        img_eq = cv2.cvtColor(img_ycrCb_eq, cv2.COLOR_YCrCb2RGB)
16        img_blur = cv2.medianBlur(img_eq, 3)
17        img_blur = img_blur.astype(np.float32)/255.0
18        processed_images.append(img_blur)
19     return processed_images

```

Min-Max Normalization

Brightness-only Histogram Equalization

Median Blurring

Agent optimized - postprocessing

```

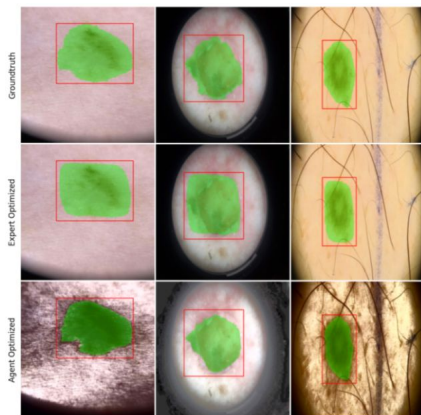
1 import torch
2 import numpy as np
3 import cv2
4 def postprocess_preds(preds):
5     target_size = (512, 512)
6     preds_copy = preds.clone().cpu().numpy()
7     N, C, H, W = preds_copy.shape
8     processed_masks = np.empty((N, 1, *target_size), dtype=np.float)
9     for idx in range(N):
10        pred_mask = preds_copy[idx, 0]
11        upsampled = cv2.resize(pred_mask, target_size, interpolation=cv2.
12        INTER_LINEAR)
13        binary = (upsampled > 0.45).astype(np.uint8)
14        kernel = cv2.getStructuringElement(cv2.MORPH_ELLIPSE, (5,5))
15        closed = cv2.morphologyEx(binary, cv2.MORPH_CLOSE, kernel)
16        processed_masks[idx, 0] = closed
17        processed_masks = torch.from_numpy(processed_masks)
18        processed_masks = processed_masks.squeeze(1)
19     return processed_masks

```

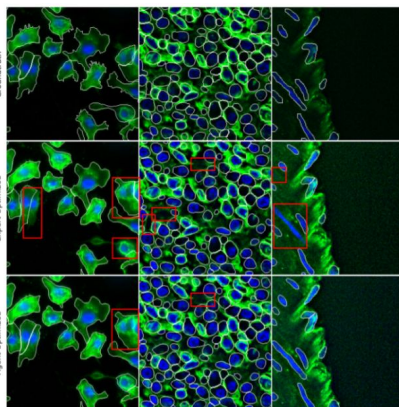
Bilinear Interpolation

Morphological Closing

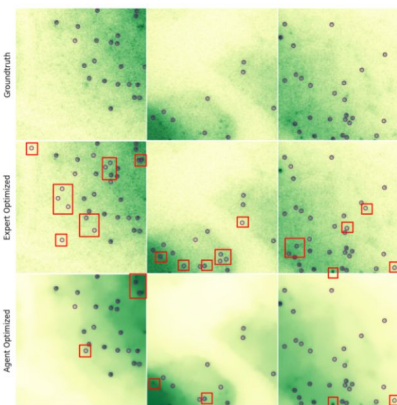
Results



Spot Detection - Polaris / DeepCell
F1 Score - Expert: 0.841 Agent: 0.929



Cell Segmentation - Cellpose 3 (cyto3)
AP @ IoU 0.5 - Expert: 0.402 Agent: 0.417



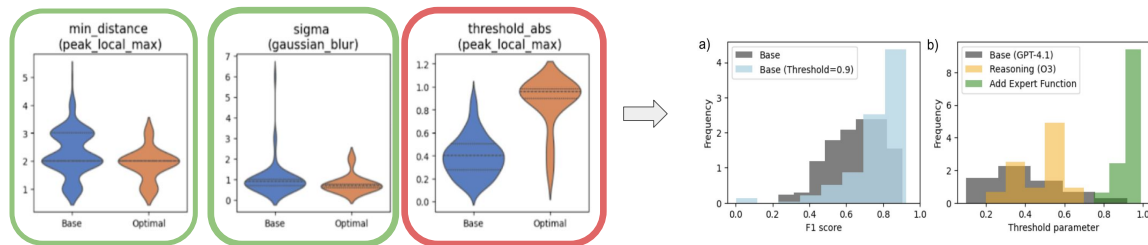
Medical Segmentation - MedSAM
NSD + DSC: Expert: 0.820 Agent: 1.037

	Expert Baseline	Base Agent
Polaris (F1)	0.841	0.867
Cellpose (AP@IoU 0.5)	0.402	0.409
MedSAM (NSD+DSC)	0.820	0.971

Add Expert Function	Add Function Bank	Reasoning LLM	Small LLM	No Data Prompt	No API List
0.929	0.889	0.844	0.805	0.856	0.868
0.410	0.416	0.412	0.397	0.406	0.417
0.888	0.943	1.020	0.918	0.952	1.037

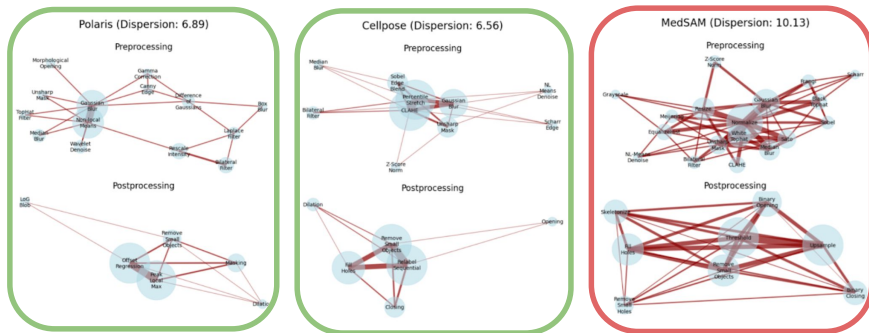
Solution Space Analysis

Parameter Space Analysis – Why does Reasoning LLM hurt Polaris performance?



- when LLM's prior on the parameter selection diverges from the optimal range, agent fails. Reasoning LLMs is more limited in its prior.

API Space Analysis – Why does Expert Function and Function Bank hurt MedSaM performance?



- When the optimal solution space is more diverse, presenting expert or prior solutions limits agent's exploration.

Search Space Summary

	API Space	Parameter Space
Polaris	Concentrated	Hard-to-optimize
Cellpose	Concentrated	Easy-to-optimize
MedSAM	Dispersed	Easy-to-optimize

Ablation Studies

Does AutoML Module help?

	Non-Agentic AutoML	Agent w/ FuncBank	Agent w/ FuncBank + AutoML
Polaris	0.844	0.889	0.877
Cellpose	0.373	0.416	0.417
MedSAM	0.879	0.943	1.014

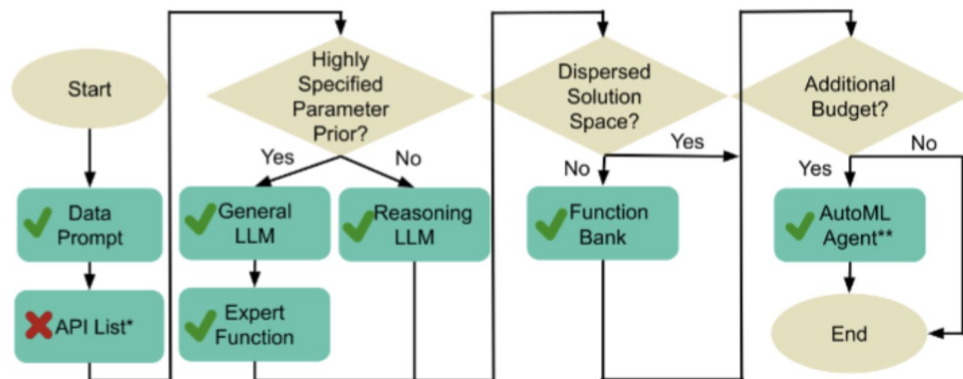
- AutoML alone is not useful.
- AutoML + Agent can be useful if not overfitting.

Does complicated Tree Search Agent help?

	Base Agent	Base Agent w/ FuncBank	AIDE Agent (Tree Search)
Polaris	0.867	0.889	0.872
Cellpose	0.409	0.416	0.414
MedSAM	0.971	0.943	0.971

- No obvious advantages from Tree Search Agent.

Insight & Impact



*Use the list only if the LLM's intrinsic knowledge is insufficient.
**Caution: extensive AutoML search might lead to overfitting.

The screenshot shows a GitHub pull request titled "Added optimized preprocessing function #95" in the repository "vanvalenlab / deepcell-spots". The pull request is merged and includes a comment from "quietpath" dated May 16. The comment describes a new image preprocessing function, `blurred_laplacian_of_gaussian()`, which outperforms existing functions on the spot detection task. It also includes a table comparing the validation and test F1 scores of the new function against a baseline.

This PR introduces a new image preprocessing function, `blurred_laplacian_of_gaussian()`, which outperforms the existing preprocessing functions on the spot detection task. This function currently only supports single-channel images, we will extend support to multi-channel images soon.

This preprocessing function was discovered by agentic superoptimization. Novel preprocessing functions were generated and tested by an LLM-based agent system using the spot detection application and the SpotNet dataset validation and test split. Our evaluation results are shown below:

	Validation (F1 Score)	Test (F1 Score)
Agent Preprocessing Function	0.7407	0.9019
Baseline Preprocessing Function	0.6640	0.8414