Agentic Superoptimization of Scientific Analysis Workflows

Anonymous Author(s)
Affiliation
Address
email

Abstract

Data-driven scientific discovery relies on complex computational workflows to process large, high-dimensional experimental datasets. However, a fundamental bottleneck exists: adapting carefully engineered computational tools to bespoke datasets studied by individual labs demands substantial manual tuning and custom code development, consuming weeks or months of expert time and slowing scientific progress. To address this bottleneck, we introduce agentic superoptimization, a new paradigm for leveraging generative AI to autonomously write customized code that can surpass human-expert-engineered solutions. We present **SciOpt**, a proof-of-concept agentic framework for superoptimizing data preparation functions directly in real-world, production-level scientific workflows, without requiring additional annotations or training. We validate our approach on challenging biology and medical imaging tasks, showing that our agent consistently outperforms expert baselines. Notably, our agent-generated code achieved was successfully deployed into a production-level scientific pipeline. Our work lays the foundation towards human-AI agent collaborative discovery in complex, real-world environments.

1 Introduction

The past decade has seen an unprecedented acceleration in data-driven scientific discovery, with researchers leveraging increasingly sophisticated computational workflows to tackle the ever-expanding volume and complexity of experimental data [27]. These recent advances, however, have revealed a fundamental bottleneck at the interface between powerful computational systems and the specific requirements of individual scientists. In almost all real-world scientific inquiry, the application of carefully engineered computational tools to new datasets (even those within the same domain) commonly demands substantial manual tuning and custom code development across all stages of the scientific workflow [26].

For instance, applying modern cell segmentation tools to novel datasets often produces poor results due to new images having reduced quality and being out-of-distribution [13], despite models being trained on increasingly diverse sets of cellular images. This issue arises due to significant variability in image acquisition between labs, including low illumination, light scattering, aberrations, differences in image resolution, and low signal-to-noise. Considerable effort is often expended in data preparation, hyperparameter tuning, and postprocessing to obtain useful results, even with modern segmentation models. These efforts can range from heuristic function development based on manual inspection of images to training custom denoising neural networks. Such bespoke engineering is a non-trivial task, and can consume months of invaluable domain expert time. This inefficiency slows the pace of scientific progress, limits the reproducibility and utility of cutting-edge computational tools, and ultimately prevents the translation of high-dimensional data into meaningful insights.

Submitted to COLM 2025 Workshop. Do not distribute.

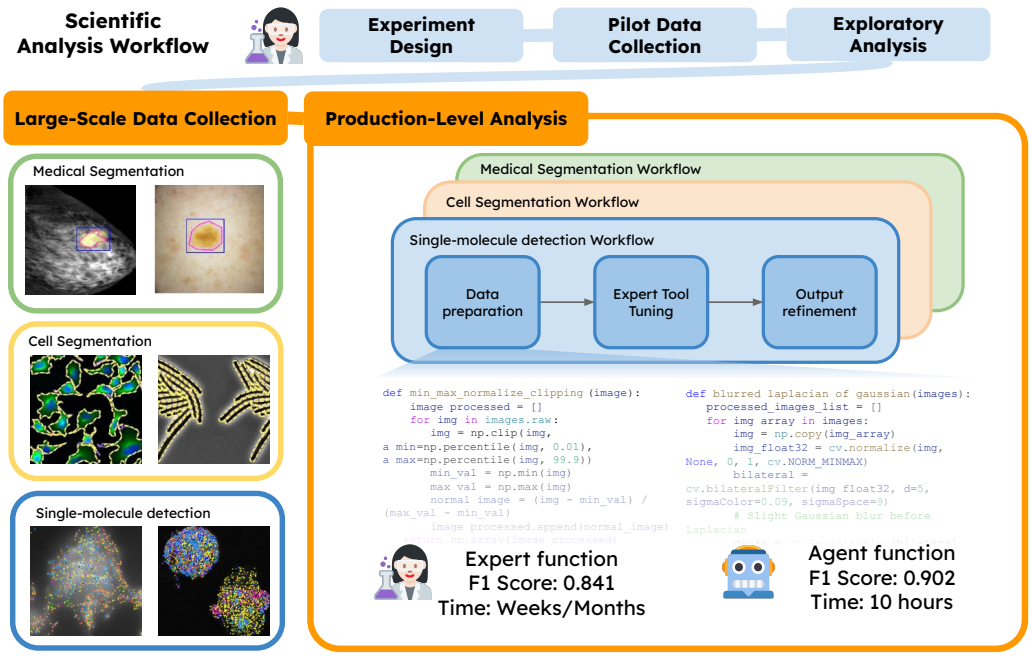

Figure 1: Overview of Scientific Analysis Workflow with Agentic Superoptimization. We consider three case settings in biomedical imaging analysis that span the multiple length scales, from single molecule to macroscopic (Section 4).

We propose agentic superoptimization of scientific analysis workflows, a new paradigm that transforms how complex scientific analyses are developed and deployed. Within this framework, superoptimization extends beyond its traditional focus on generating maximally optimized (e.g., fastest or smallest) code, to achieving scientific analysis outcomes that surpass human-expert-engineered solutions. This unique combination of advanced agentic capabilities with the objective of surpassing expert-level performance in integrated real-world production-level scientific workflows distinguishes our work from existing agentic systems that typically focus on exploratory data analysis [20], task automation [3], or program correctness [1].

We developped **SciOpt**, a proof-of-concept agent framework to evaluate the effectiveness of agentic superoptimization. Our experiments demonstrate that these agents can indeed develop solutions that outperform expert-designed workflows. We studied effective agent designs: surprisingly, common strategies such as the inclusion of a function bank can inadvertently restrict agent performance. Understanding and navigating the design choices presents unique and demanding challenges for agentic algorithms and systems, thereby pushing the boundaries of what LLM agents can achieve in complex, real-world environments.

We additionally demonstrate the real-world applicability of agentic superoptimization, through the successful deployment of agent-generated functions into a real-world production pipeline (Figure 4). Tackling the pervasive workflow adaptation challenge holds considerable significance in science, by automating painstaking manual code adaptations. Looking forward, while the present work investigates agentic superoptimization in biomedical imaging analysis, this paradigm has broad potential application to further accelerate scientific discovery.

2 Problem Statement

We address the problem of superoptimizing scientific analysis workflows, as depicted in Figure 1. Given a production-level analysis tool for biomedical imaging, the objective is to develop and configure components of workflows to maximize performance on a targeted benchmark. In particular, one needs to write programs to instantiate such workflows, resulting in a combinatorially (or infinitely) large search space.

Superoptimization. The term superoptimization comes from the program synthesis community [22] - given a reference function π_{ref}, the goal is to find a program π in a programming language

$\mathcal{L}$ that optimizes a score function $S(\pi) \in \Re$ such that $S(\pi) > S(\pi_{ref})$. Prior superoptimization settings include inductive program synthesis, where the score function is typically defined as compute or memory cost of the program on a specific hardware or execution environment, conditioned on consistency with respect to a specification (e.g., consistent with unit tests) [24, 23, 22].

Because the space of programs is combinatorially (or infinitely) large, it is important to regularize the optimization problem to some notion of "simple" or "natural programs". This naturally leads to the following maximum a posteriori (MAP) estimation problem [4]:

$$\pi^\star = \arg\max_\pi p_{\mathcal{L}}(\pi|S) = \arg\max_\pi \underbrace{p_{\mathcal{L}}(S|\pi)}_{\text{By Execution}} \cdot \underbrace{p_{\mathcal{L}}(\pi)}_{\text{By Generation}} , \tag{1}$$

where $p_{\mathcal{L}}(S|\pi)$ expresses the likelihood of a program execution to achieve a certain score, and $p_{\mathcal{L}}(\pi)$ is a prior distribution on "natural" programs. In practice, we will use a high-quality large language model (LLM) to sample from $p_{\mathcal{L}}(\pi)$. The term "superoptimization" refers to solving (1) to a superior level than what is practically achievable by human experts.

Superoptimization for Scientific Data Analysis. We model scientific data analysis as a superoptimization problem with three task-specific inputs: (1) a domain-specific dataset of input-output examples $\mathcal{D} := \{(\mathbf{x}_i, \mathbf{y}_i)\}_{i=1}^N$ and a score function $S_{\mathcal{D}}$ defined on the dataset; (2) an execution environment that includes a production-quality analysis tool such as those depicted in Figure 1; (3) a natural language task description with relevant domain information provided by a domain expert.

All these inputs are either naturally expressible in natural language or can be compiled into executable programs with interpretable natural language meaning. This enables leveraging LLMs to capture the semantic meaning and use it to guide the superoptimization process. LLMs approach the program generation problem as a token prediction problem, directly approximating the program likelihood by training on internet-scale datasets. That is, $p_{\mathcal{L}}(\pi|S_{\mathcal{D}}) \approx p_{\text{LLM}}(\langle\pi\rangle|\langle\mathcal{L}\rangle; \langle S_{\mathcal{D}}\rangle)$, where $\langle\cdot\rangle$ represents a natural language serialization of the respective variables. Because current LLMs cannot reliably predict how their generated code will execute [19], our synthesizer incorporates an agentic workflow [29, 28]: it runs the code, evaluates the output, and feeds the result back to the model. The next sections describe this workflow in detail.

3 Agentic AI framework

To demonstrate the task of superoptimizing scientific analysis workflows, we designed a proof-of-concept agent system, incorporating the design choices commonly seen in the field, including a dual-agent generation-execution feedback loop [7, 30] and a function bank [21, 17]. The key inputs for our framework are: dataset-specific prompts, task-specific prompts, and a library of primitives that act as building blocks for generated functions.

Overall Architecture. The dual-agent feedback loop in Figure 2 combines a code-writing agent that synthesizes new functions based on its prompting and an execution agent to evaluate the functions within a complex workflow and dataset. Execution feedback and evaluation scores are sent back to the code-writing agent to guide revisions to the generated code, forming the feedback loop. The function bank stores generated functions and their associated execution results. Sampling previous functions and inserting it into the code-writing agent's context can guide later code generations. Although many of these components have been utilized in prior efforts, their specific contributions to achieving superoptimization in scientific analysis workflows remain to be fully elucidated. We investigate this question in the present work.

Prompt Structure. The prompt for the coding agent is structured into four key components: data description, task specification, available primitives, and function samples. The data description details the characteristics of the specific dataset, including attributes such as image type (e.g., RGB or grayscale) and data modality (e.g., dermoscopy, X-ray, or spatial transcriptomics). The task specification outlines the target task, relevant evaluation metrics, and the broader workflow context. Since we are focused on biomedical imaging tasks, the available primitives consist of the OpenCV API functions that the agent can utilize to generate new functions.

Optimization Procedure. The optimization procedure unfolds in two phases. During Phase 1, the function bank remains unused to encourage the coding agent's exploratory generation of diverse

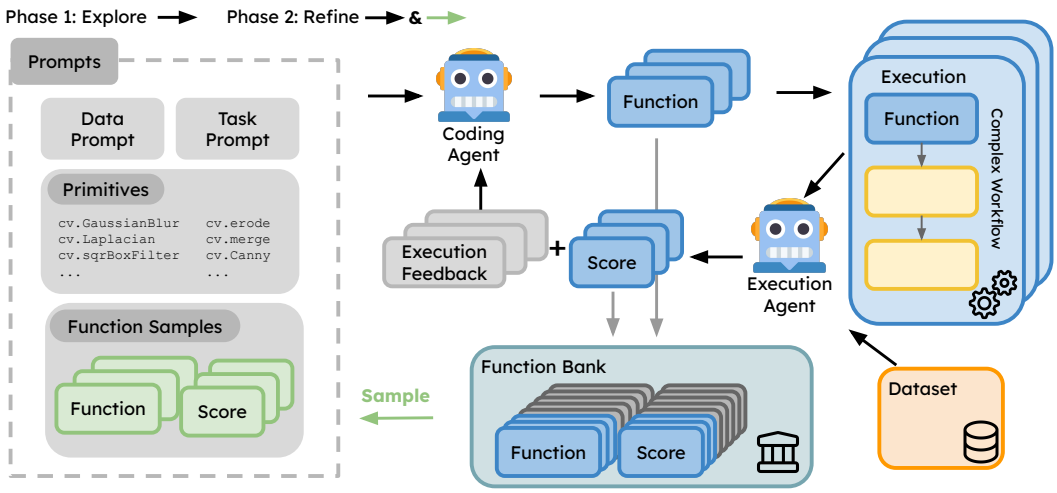

Figure 2: **Agent Framework for Superoptimization**. The key components include a dual-agent generation-execution feedback loop and a function bank.

functions. In Phase 2, we strategically sample functions from the bank and incorporate this historical information into the prompt, guiding the coding agent to further refine its function generation based on prior knowledge. The function samples include the highest-performing and lowest-performing functions from the function bank, along with their corresponding performance scores.

Sampling. In each iteration, the coding agent generates multiple functions using the same conversation context. This design choice explicitly motivates the agent to create independent and different functions from the same dependent view of the previous function bank, which implicitly encourages the exploration of a broader functional space. We also run multiple independent rollouts with different random seeds.

Remark on Bottlenecks. Because the tools we optimize for (see Section 4) are computationally intensive scientific analysis software, the entire process is typically bottlenecked by tool calls rather than LLM queries. As such, there is less incentive to use faster but less performant LLMs.

4 Case Studies

We consider three biomedical imaging applications as case studies. A production-quality domain-specific tool is provided as part of the workflow execution, together with a domain expert engineered preprocessing function as the baseline, as well as a labeled dataset for optimizing the workflow. The three case studies span multiple length scales, from single molecule to cellular to macroscopic.

4.1 Polaris: Single Molecule Spot Detection

We focused on sub-pixel fluorescent spot detection for image-based spatial transcriptomics (Figure 1, blue box), evaluating only the spot detection component of the Polaris pipeline [12]. The dataset used for this task consists of the held-out validation and test splits from the original training data. All images are from various modalities using different RNA capturing and tagging methods. The expert baseline preprocessing applied involved intensity normalization with clipping. The optimization objective for this task was the maximization of the F1 score.

4.2 Cellpose: Cell Segmentation

The objective of this task is cell instance segmentation on multiple modalities such as whole cell & nucleus (Figure 1, yellow box). The tool we use is the 'cyto-3' model from Cellpose3 [25], a U-Net based network pre-trained for general cell segmentation. For this case study, we curated a publicly available and reconstructable subset from the reported Cellpose3 dataset, including test sets from the Cellpose3 dataset release (68 images), improved TissueNet 1.1 test set (1324 images), Omnipose fluorescent bacterial test set (75 images), and Omnipose phase-contrast bacterial test set (148 images).

Datasets involving complex mask corrections were excluded. The expert baseline applies per-channel percentile-based min-max normalization. The metric used for optimization and evaluation is the average precision at an Intersection over Union (IoU) threshold of 0.5.

4.3 MedSAM: Medical Segmentation

This task involves medical image segmentation (Figure 1, green box). The tool we use is MedSAM [16], which is an extension of the Segment Anything Model [10] (SAM) specifically adapted for medical imaging domains, and trained on over 1.5 million image-mask pairs. For this case study, we used randomly sampled subsets of the publicly released validation set from the MedSAM Codabench competition [15], focusing on two specific modalities: Dermoscopy (RGB) and X-ray (grayscale). Each image type required specific preprocessing: RGB Dermoscopy images used min-max intensity scaling, while grayscale X-ray images (converted to 3 channels) underwent intensity clipping before rescaling. Evaluation was performed on the full generated test sets for each modality. The optimization and evaluation objective was to maximize the sum of the Normalized Surface Dice (NSD) and Dice Similarity Coefficient (DSC) scores.

5 Experiments

We now present experimental validation on our three case studies. We first demonstrate that agentic superoptimization is indeed achievable using our framework (Section 5.2). We then detail our empirical investigations into the specific design choices influencing the agent's performance, measured by scoring the top performing functions (Section 5.3). We show how inductive biases—originating from the agent's pretraining, prompting strategy, and reliance on function bank history data—can constrain its exploration and performance. To further study the impact of a wider solution space, we also incorporate results from exhaustive searching using a non-LLM-based conventional optimization approach (AutoML) (Section 5.4). Finally, we compare the agent's performance and exploration characteristics when based on different underlying LLMs, highlighting the considerable effect that LLMs' inherent biases have on the search process and final outcomes (Section 5.5).

5.1 Experimental Setup

Agent optimization experiments were conducted on the three case studies described in Section 4: Polaris, Cellpose, and MedSAM. For each task, 20 independent rollouts were performed, with each rollout consisting of 20 iterations. In experiments where the function bank was utilized, it was introduced at the 5th iteration, with top 3 and bottom 3 functions being added to the prompt. We aggregated and selected the final top $K = 1, 10$ functions, evaluated them on the test set, and reported the highest scores. All standard experiments use GPT-4.1 as the base LLM for the code-writing agent.

	Expert	**Full System**	
		$K = 1$	$K = 10$
Polaris (F1)	0.841	**0.902**	**0.902**
Cellpose (AP@IoU 0.5)	0.403	**0.410**	**0.410**
MedSAM, Dermo (NSD+DSC)	0.836	**0.846**	**0.881**
MedSAM, X-ray (NSD+DSC)	0.750	**0.770**	**0.776**

Table 1: Main results on test set. Comparing performance with function selection thresholds $K = 1$ and $K = 10$.

5.2 Agentic Superoptimization vs. Expert Baselines

The main question we ask is: can our agent write code for scientific analysis workflows that outperforms expert-written code. This is a non-trivial task given the complexity of the tools used, and the fact that expert baselines are designed over a span of weeks or months.

Our results in Table 1 definitively provide a positive answer to this question. Across all case studies, our AI Agent is able to produce workflows that outperform the expert-written baselines on the test set. The minimal improvement of Top-10 performance over Top-1 suggests that the best performing solution on the validation set tends to be the best on the test set as well. Figure 3 shows the learning curve on the validation set, which shows potential for further improvement with more iterations.

Figure 4 shows an example function written by our agent for the Polaris spot detection setting. Compared to the expert-written baseline preprocessing function, which achieves an F1 score of 0.664

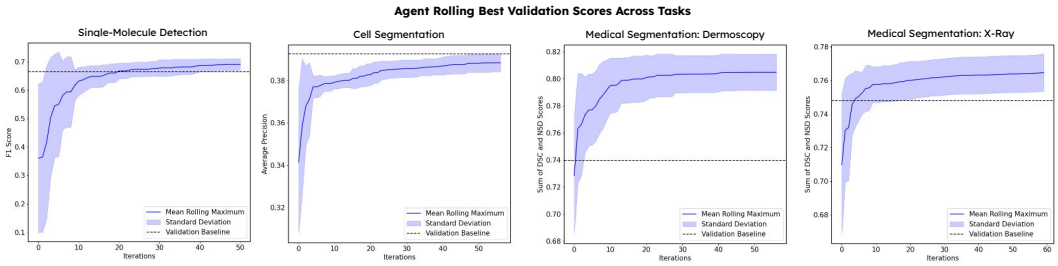

Figure 3: Depicting learning curves on validation set performance. The dotted horizontal line is the validation performance of the expert baseline. We see that average performance continues to improve even when we terminate the experiments, suggesting further improvements are possible.

	Full System		**−Function Bank**		**−Task Prompts**		**−Data Prompts**	
	$K = 1$	$K = 10$	$K = 1$	$K = 10$	$K = 1$	$K = 10$	$K = 1$	$K = 10$
Polaris (F1)	**0.902**	0.902	0.870	**0.926**	0.786	0.925	0.908	0.908
Cellpose (AP@IoU 0.5)	0.410	0.410	**0.412**	**0.412**	0.405	0.405	0.410	0.410
MedSAM, Dermo (NSD+DSC)	0.846	0.881	**0.932**	**0.932**	0.912	0.912	0.918	0.925

Table 2: Test set results on ablations of design choices, comparing $K = 1$ and $K = 10$ thresholds.

on validation and 0.841 on test, the framework-generated preprocessing function achieves an F1 score of 0.741 on validation and 0.902 on test. An improvement on both validation and test indicates that the agent identified more complex image processing methods that better align with the distribution and noise of the dataset than the expert-written baseline. Notably, the agent-generated code in Figure 4 employs an unusual combination of OpenCV's Gaussian blur and Laplacian operator.

We re-emphasize that these case studies involve production-quality software workflows. For instance, the function in Figure 4 is currently being integrated into the Polaris production pipeline.

5.3 Impact of design choices

We conduct an array of experiments to understand which agentic design choices result in superior superoptimization. Due to computational limitations, the MedSAM X-Ray task was excluded from this analysis. The **Full System** configuration was compared against the following variants:

No Function Bank: Each iteration was provided the same static context, providing no historical information from generated functions and scores.

No Task Prompt: We omitted descriptions about tasks (e.g. "segmentation", "detection"), domain tools, or any associated metric descriptors.

No Data Prompt: We removed any references to the nature of the data such as "medical," "X-Ray," or any domain-specific terminology.

Surprisingly, many variants do not degrade performance compared to the default full system setting, as shown in Table 2. Most notably, removing the function bank consistently led to improved performance. We found that removing task-specific or data-specific prompts generally improved predictive performance for MedSAM and Polaris. Conversely, with Cellpose, removing the task-specific prompt decreased performance, while data-prompts had no effect. These results highlight the need for rigorous design and empirical evaluation of AI agent innovations, as they may not actually improve performance, particularly real-world settings such as optimizing production-quality workflows.

5.4 Comparison with AutoML

A common alternative is to use AutoML methods, which effectively amount to some form of random search or Bayesian optimization. Making a completely fair comparison is difficult because the setup in AutoML requires a human expert to specify the allowable function primitives in a more rigid and

```python
def blurred laplacian of gaussian(images):
   processed_images_list = []
   for img_array in images:
       img = np.copy(img_array)
       img_float32 = cv.normalize(img, None, 0, 1,
cv.NORM_MINMAX).astype(np.float32)
       bilateral = cv.bilateralFilter(img_float32, d=5,
sigmaColor=0.09, sigmaSpace=9)
       gauss = cv.GaussianBlur(bilateral, (3,3), 0)
       lap = cv.Laplacian(gauss, cv.CV_32F, ksize=3)
       abs_lap = np.abs(lap)
       lap_norm = cv.normalize(abs_lap, None, 0, 1,
cv.NORM_MINMAX).astype(np.float32)
       if img_array.ndim == 3 and img_array.shape[2] == 1:
           lap_norm = lap_norm[:, :, np.newaxis]
       processed_images_list.append(lap_norm)
   return np.array(processed_images_list, dtype=np.float32)
```

Figure 4: An agent-generated Polaris preprocessing function, to be integrated into production.

time-consuming way compared to our agentic prompting. The analysis here is meant to provide a sense of the reliability of each approach.

Setup We compared our agentic framework to standard AutoML using Optuna [2] for hyperparameter optimization. We organized OpenCV functions into families with defined parameter search ranges, allowing 2-4 primitive operations, and optimized using the Tree-structured Parzen Estimator (TPE) sampler. Given the results in Section 5.3, we select the best performing agent variant for this comparison.

Results Following an extensive search of the manually defined optimization space, AutoML's best function sometimes achieves performance on par with our agent's best functions ($K = 1$ and $K = 10$) across tasks, but not reliably so. Notably, the performance of AutoML was particularly poor on tasks exhibiting less saturated performance ceilings, such as Cellpose where AutoML underperformed the expert-written baseline, suggesting greater potential for agentic superoptimization in these scenarios. Moreover, it might be interesting future work to merge these two systems to inherit the strengths of both.

	Agent Best		**AutoML**
	$K = 1$	$K = 10$	
Polaris (F1)	0.902	**0.926**	0.916
Cellpose (AP@IoU 0.5)	**0.412**	**0.412**	0.339
MedSAM, Dermo (NSD+DSC)	0.932	0.932	**0.936**

Table 3: Comparing Agent system with AutoML on test performance. We selected the best performing variant from Table 2 based on validation performance.

5.5 Impact of LLM choice

Our code-writing LLM replacement ablation demonstrated that the choice of model significantly influences the pre-training inductive bias manifested in code generation. Specifically, the LLaMA 3.3 70B model exhibited poor exploration of the primitive-defined search space and a tendency to generate repetitive functions across tasks, irrespective of context. This limitation resulted in worse performance compared to the agent employing a GPT-4.1 backbone. Moreover, since the overall system is bottlenecked by the tool call (e.g., Polaris, Cellpose, MedSAM), there is little incentive to use faster but less performant LLMs.

	GPT-4.1		**LLaMA 3.3 70B**	
	$K = 1$	$K = 10$	$K = 1$	$K = 10$
Polaris	**0.902**	**0.902**	0.881	0.888
Cellpose	**0.410**	**0.410**	0.404	0.404
MedSAM, Dermo	0.846	**0.881**	0.821	0.865

Table 4: Results comparing LLM performance

6 Related Work

Program Synthesis & Superoptimization. Recent years have seen a proliferation of systems designed to exceed human performance on specialized coding tasks. AlphaCode solves competition-level programming problems [14], while AlphaTensor and AlphaDev produce super-optimized matrix-multiplication and sorting routines, respectively [5, 18]. The unifying thread in these methods is the combination of large, publicly sourced pre-training datasets with carefully chosen inductive biases to make optimization faster. In superoptimization, [24] demonstrated that an LLM can suggest performance-improving edits in low-level code. Our approach, instead, focuses on superoptimizing scientific workflows written in high-level languages. Furthermore, instead of training a specialized model to suggest code edits, our approach leverages the code generation abilities of a pretrained LLM as well as its domain-specific knowledge to identify optimizations in scientific workflows while following expert domain specific instructions. Recently, Funsearch [21] demonstrates that the code generated with pretrained LLMs can be optimized with simple evolutionary optimization to discover novel and highly performant heuristics (expressed in code) to solve problems in combinatorial mathematics. Our work is similar to Funsearch in that it leverages the code generation and domain-specific knowledge of an LLM. In contrast, we propose an agentic optimization loop to discover optimal or near-optimal data preprocessing algorithms for large-scale scientific workflows.

AutoML. Automated Machine Learning (AutoML) comprises a wide variety of techniques for automating end-to-end machine learning pipelines, including model hyperparameter optimization [11, 2], feature engineering [9], neural network topology [31, 8], and data cleaning and preprocessing [11, 6, 2]. Frameworks like Optuna [2] enable sophisticated hyperparameter search over user-defined spaces. Constructing such detailed search spaces, however, demands considerable effort and requires specialized knowledge of the intricacies of such operations. This stands in contrast to our agentic system, which eliminates the need for such labor-intensive manual configuration.

Scientific Analysis Agent. Recent work applying LLM agents to scientific analysis and discovery have found success and shown efficiency gains over human experts. While DiSciPLE [17] demonstrates interpretable program learning for new scientific tasks, outperforming human experts within a 1.5-hour limit, our work focuses on agentic optimization of production-level, open-source tools in established scientific domains. This contrasts with Popper [7], which, though effective for short-duration hypothesis falsification (under one hour), only matched PhD-level performance. Our superoptimization pipeline, however, is designed to surpass human expert baselines that have been refined over much longer periods (weeks to months).

7 Discussion

Limitation. Our current framework, while promising for superoptimizing scientific workflows, has limitations. The LLM's direct generation of primitive function hyperparameters is inefficient and unreliable; future work should integrate a lightweight hyperparameter optimizer. Additionally, we must address inherent LLM pretraining biases. Our empirical study is currently limited to image preprocessing in three biomedical tasks. Future work will expand the primitive library and explore diverse domains, significantly broadening our impact.

Conclusion. We introduce agentic superoptimization, a novel paradigm leveraging generative AI to address the critical and time-consuming bottleneck of adapting computational tools to bespoke datasets in scientific analysis. Our AI agent framework demonstrates the ability to autonomously generate and refine pre-processing code for real-world bio/medical imaging workflows, successfully surpassing expert-engineered baselines. Through systematic experimentation and ablation studies, we further identified that agent performance is significantly influenced by specific design choices. The successful integration of agent-derived solutions into production-level workflows highlights the practical viability of this approach. Agentic superoptimization accelerates scientific innovation by automating workflow adaptations and expands LLM agent capabilities in real-world applications. We encourage future research at this intersection, extending agentic superoptimization to other stages of the scientific workflow, building the future of human-AI collaborative scientific discovery.

References

[1] Pooja Aggarwal, Oishik Chatterjee, Ting Dai, Prateeti Mohapatra, Brent Paulovicks, Brad Blancett, and Arthur De Magalhaes. Codesift: An llm-based reference-less framework for automatic code validation. In *2024 IEEE 17th International Conference on Cloud Computing (CLOUD)*, pages 404–410, 2024.

[2] Takuya Akiba, Shotaro Sano, Toshihiko Yanase, Takeru Ohta, and Masanori Koyama. Optuna: A next-generation hyperparameter optimization framework. In *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, KDD '19, page 2623–2631, New York, NY, USA, 2019. Association for Computing Machinery.

[3] Noel Crawford, Edward B. Duffy, Iman Evazzade, Torsten Foehr, Gregory Robbins, Debbrata Kumar Saha, Jiya Varma, and Marcin Ziolkowski. Bmw agents – a framework for task automation through multi-agent collaboration, 2024.

[4] Kevin Ellis, Catherine Wong, Maxwell Nye, Mathias Sable-Meyer, Luc Cary, Lucas Morales, Luke Hewitt, Armando Solar-Lezama, and Joshua B. Tenenbaum. Dreamcoder: Growing generalizable, interpretable knowledge with wake-sleep bayesian program learning, 2020.

[5] Alhussein Fawzi, Matej Balog, Aja Huang, Thomas Hubert, Bernardino Romera-Paredes, Mohammadamin Barekatain, Alexander Novikov, Francisco J R. Ruiz, Julian Schrittwieser, Grzegorz Swirszcz, et al. Discovering faster matrix multiplication algorithms with reinforcement learning. *Nature*, 610(7930):47–53, 2022.

[6] Matthias Feurer, Katharina Eggensperger, Stefan Falkner, Marius Lindauer, and Frank Hutter. Auto-sklearn 2.0: Hands-free automl via meta-learning. *arXiv:2007.04074 [cs.LG]*, 2020.

[7] Kexin Huang, Ying Jin, Ryan Li, Michael Y. Li, Emmanuel Candès, and Jure Leskovec. Automated hypothesis validation with agentic sequential falsifications, 2025.

[8] Haifeng Jin, François Chollet, Qingquan Song, and Xia Hu. Autokeras: An automl library for deep learning. *Journal of Machine Learning Research*, 24(6):1–6, 2023.

[9] James Max Kanter and Kalyan Veeramachaneni. Deep feature synthesis: Towards automating data science endeavors. In *2015 IEEE International Conference on Data Science and Advanced Analytics (DSAA)*, pages 1–10, Campus des Cordeliers, Paris, France, October 2015. IEEE.

[10] Alexander Kirillov, Eric Mintun, Nikhila Ravi, Hanzi Mao, Chloe Rolland, Laura Gustafson, Tete Xiao, Spencer Whitehead, Alexander C. Berg, Wan-Yen Lo, Piotr Dollár, and Ross Girshick. Segment anything. *arXiv:2304.02643*, 2023.

[11] Brent Komer, James Bergstra, and Chris Eliasmith. Hyperopt-sklearn. In Frank Hutter, Lars Kotthoff, and Joaquin Vanschoren, editors, *Automated machine learning: Methods, systems, challenges*, pages 97–111. Springer International Publishing, Cham, 2019.

[12] Emily Laubscher, Xuefei Wang, Nitzan Razin, Tom Dougherty, Rosalind J Xu, Lincoln Ombelets, Edward Pao, William Graf, Jeffrey R Moffitt, Yisong Yue, et al. Accurate single-molecule spot detection for image-based spatial transcriptomics with weakly supervised deep learning. *Cell Systems*, 15(5):475–482, 2024.

[13] Kwanyoung Lee, Hyungjo Byun, and Hyunjung Shim. Cell segmentation in multi-modality high-resolution microscopy images with cellpose. In *Proceedings of The Cell Segmentation Challenge in Multi-modality High-Resolution Microscopy Images*, volume 212 of *Proceedings of Machine Learning Research*, pages 1–11. PMLR, 28 Nov–09 Dec 2023.

[14] Yujia Li, David Choi, Junyoung Chung, Nate Kushman, Julian Schrittwieser, Rémi Leblond, Tom Eccles, James Keeling, Felix Gimeno, Agustin Dal Lago, et al. Competition-level code generation with alphacode. *Science*, 378(6624):1092–1097, 2022.

[15] Jun Ma. Cvpr 2024: Segment anything in medical images on laptop. In *CVPR 2024 Workshop*. OpenReview.net, 2024.

[16] Jun Ma, Yuting He, Feifei Li, Lin Han, Chenyu You, and Bo Wang. Segment anything in medical images. *Nature Commun.*, 15(1):654, January 2024.

[17] Utkarsh Mall, Cheng Perng Phoo, Mia Chiquier, Bharath Hariharan, Kavita Bala, and Carl Vondrick. Disciple: Learning interpretable programs for scientific visual discovery, 2025.

[18] Daniel J Mankowitz, Andrea Michi, Anton Zhernov, Marco Gelmi, Marco Selvi, Cosmin Paduraru, Edouard Leurent, Shariq Iqbal, Jean-Baptiste Lespiau, Alex Ahern, et al. Faster sorting algorithms discovered using deep reinforcement learning. *Nature*, 618(7964):257–263, 2023.

[19] Ansong Ni, Miltiadis Allamanis, Arman Cohan, Yinlin Deng, Kensen Shi, Charles Sutton, and Pengcheng Yin. Next: Teaching large language models to reason about code execution, 2024.

[20] Zeeshan Rasheed, Muhammad Waseem, Aakash Ahmad, Kai-Kristian Kemell, Wang Xiaofeng, Anh Nguyen Duc, and Pekka Abrahamsson. Can large language models serve as data analysts? a multi-agent assisted approach for qualitative data analysis, 2024.

[21] Bernardino Romera-Paredes, Mohammadamin Barekatain, Alexander Novikov, Matej Balog, M Pawan Kumar, Emilien Dupont, Francisco JR Ruiz, Jordan S Ellenberg, Pengming Wang, Omar Fawzi, et al. Mathematical discoveries from program search with large language models. *Nature*, 625(7995):468–475, 2024.

[22] Raimondas Sasnauskas, Yang Chen, Peter Collingbourne, Jeroen Ketema, Jubi Taneja, and John Regehr. Souper: A synthesizing superoptimizer. 2017.

[23] Eric Schkufza, Rahul Sharma, and Alex Aiken. Stochastic superoptimization. In *ACM SIGARCH computer architecture news*, volume 41, pages 305–316. ACM, 2013. Number: 1.

[24] Alexander Shypula, Aman Madaan, Yimeng Zeng, Uri Alon, Jacob Gardner, Milad Hashemi, Graham Neubig, Parthasarathy Ranganathan, Osbert Bastani, and Amir Yazdanbakhsh. Learning performance-improving code edits, 2024.

[25] Carsen Stringer and Marius Pachitariu. Cellpose3: one-click image restoration for improved cellular segmentation. *Nature Methods*, pages 1–8, 2025.

[26] Ana Trisovic, Matthew K. Lau, Thomas Pasquier, and Mercè Crosas. A large-scale study on research code quality and execution. *Scientific Data*, 9(1):60, February 2022. Publisher: Nature Publishing Group.

[27] Hanchen Wang, Tianfan Fu, Yuanqi Du, Wenhao Gao, Kexin Huang, Ziming Liu, Payal Chandak, Shengchao Liu, Peter Van Katwyk, Andreea Deac, et al. Scientific discovery in the age of artificial intelligence. *Nature*, 620(7972):47–60, 2023.

[28] John Yang, Carlos E. Jimenez, Alexander Wettig, Kilian Lieret, Shunyu Yao, Karthik Narasimhan, and Ofir Press. Swe-agent: Agent-computer interfaces enable automated software engineering, 2024.

[29] Shunyu Yao. *Language Agents: From Next-Token Prediction to Digital Automation*. PhD thesis, Princeton University, 2024.

[30] Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik Narasimhan, and Yuan Cao. React: Synergizing reasoning and acting in language models. In *International Conference on Learning Representations (ICLR)*, 2023.

[31] Lucas Zimmer, Marius Lindauer, and Frank Hutter. Auto-pytorch tabular: Multi-fidelity metalearning for efficient and robust autodl. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, pages 3079 – 3090, 2021. also available under https://arxiv.org/abs/2006.13799.