Benchmarking Large Language Models for Bio-Image Analysis Code Generation

Paper #363

Abstract. In the computational age, life-scientists often solve scientific bio-image analysis (BIA) questions using Python code, even though many of them are not trained in programming. As a common use-case for Large Language Models (LLMs) is code-generation, we see potential in using LLMs for BIA. We present a quantitative benchmark to estimate the capability of LLMs to generate code for solving common BIA tasks. Our benchmark consists of 57 human-written prompts, corresponding reference solutions, written in Python, and unit-tests to evaluate functional correctness of potential solutions. We demonstrate the benchmark by comparing 6 LLMs. We propose mid-/long-term efforts in maintaining and extending the benchmark by the BIA open-source community to ensure community needs are covered properly. This way we can support users when deciding for an LLM and also guide LLM developers in moving the frontier of current capabilities of LLMs towards more advanced applications in BIA.

1 Introduction

Many projects in biology involve state-of-the-art microscopy and quantitative bio-image analysis (BIA), which often requires programming skills. As programming is commonly not taught to life-scientists, we see potential in using large language models for this task. Since modern Large Language Models (LLMs) such as chatGPT (OpenAI et al. 2023) were introduced, they change how humans interact with computers. LLMs were originally developed to solve natural language processing tasks such as text classification, language translation, or question answering. Interestingly, these models are also capable of translating human languages, such as English, into programming languages, such as Python. Moreover, they can produce executable code that solves tasks defined by human natural language input [4]. This capability has huge potential for interdisciplinary research areas such as microscopy bio-image analysis [16]. LLMs can fill a gap where scientists with limited programming skills meet image analysis tasks which often require knowledge of programming languages. LLMs are indeed capable of writing BIA code as demonstrated in [17], but it is yet unclear where the limitations of this technology are in the BIA context. Multiple LLM code generation benchmarks have been proposed [5, 2, 13, 21, 9]. Thus, we think the bioimaging community urgently needs an openly accessible, quantitative way to measure those capabilities in particular given that LLM technology is also developing rapidly. We present such a benchmark, based on HumanEval [5], an established code-generation benchmark but ours is tailored for scientific questions in the Bio-Image Analysis context.

2 Methods

Our proposed benchmark consists of 55 human-written Python functions which contain documentation about what the function is supposed to do. An example is shown in Figure 1. We kept the documentation intentionally short, because we intend to use LLMs to facilitate coding for bio-image analysts. This short documentation, the docstring, together with the function signature is passed to an LLM as part of a prompt asking to complete the code. The human-written implementation function body is not passed to the LLM, and just serves as a reference solution. Additionally, our benchmark provides unit-tests for these python functions to evaluate their functional correctness. If the code generated by the language model is executable and produces results which pass the unit-tests, this LLM is considered to be able to solve the problem described in the documentation of our function. Every prompt is sent multiple times to the LLM and we measure how often the generated code passes the tests. Here, we follow the established standard pass@k [5], which estimates the probability that, if asked k times, the LLM will at least once give the correct answer. We particularly focus on the practically most relevant special case pass@1 , i.e. we want to know how likely it is that the first generated solution works. Our selected prompts range from basic image analysis tasks, such as applying an edge-preserving denoising filter to an image, over intermediate tasks such as labeling objects in a binary image and counting them, shown in Figure 1, to challenging workflows including image processing steps, descriptive statistics, tabular data wrangling and dimensionality reduction. There is also a positive-control test-case, called return_hello_world, which is intentionally kept very simple to test if a specific LLM model is capable of solving a trivial base task at all. A list of all test-cases and corresponding docstrings is given in Supplementary List 1. To enable future extension and reproducing our benchmark, we provide the necessary infrastructure to turn the list of test-case Jupyter Notebooks into a JSONL file that has the same format as the evaluation framework of HumanEval [5]. We also did minor modifications to this framework to be able to execute the benchmark for our purposes. For example, we added code that moves example images to the temporary folder from which the test-case code is executed. That way our benchmark can contain operations accessing files and folders, which the original HumanEval benchmark was not capable of. All modifications are explained in our Github repository and the supplementary Zip file. We demonstrate the benchmark by comparing the capabilities of a range of state-of-the-art LLMs covering commercial and freely available or open source models: gpt-3.5-turbo-1106, gpt-4-1106-preview, gpt-4-2024-04-09, codellama [18], claude-3-opus-20240229 [1], gemini-pro [19]. Code for benchmarking gemini-1.5-pro and gemini-ultra are available as well, but could not be executed due to rate limits. Code for benchmarking Mistral-7B-Instruct-v0.2

[12] via the Blablador infrastructure of Helmholtz AI is also available. This model was not benchmarked due to technical difficulties at the service provider at the time of benchmarking. For benchmarking the models, we generated 10 code samples of the 57 test-cases of the 6 models. Benchmarking was done on a Windows 10 Laptop with an AMD Ryzen 9 6900 CPU, 32 GB of RAM and a NVidia RTX 3050 TI GPU with 4 GB of RAM. Codellama, the only locally executed model in our selection, was accessed via ollama version 0.1.29 [15]. The gemini-pro model was accessed via the Google Vertex API [10], which did not support specifying a version. Thus, we document here that the benchmark was executed on April 16th and 17th 2024. We also summarize used libraries in the generated code and resulting error messages by counting libraries names and common error messages observed in the results of the code evaluation step. All test-cases as human-readable Jupyter Notebooks and packaged as JSONL file, sampling and evaluation code, generated samples and data analysis/visualization notebooks are available open-source in the github repository of the project. All respective Python package versions are documented in the environment.yml file in the Github repository and the supplementarty material facilitating full reproducibility of our analysis.

Figure 1. The example test-case label_binary_image_and_count_labels is implemented as Jupyter Notebook consisting of a function signature consuming a binary image and a docstring describing what the function is supposed to do. These two serve as part of a prompt to the LLM asking to complete the code. The function body serves as a reference solution (sometimes referred to as canonical solution), which allows writing a unit test. If the unit test passes for the code generated by an LLM, indicating functional correctness, the LLM is defined as capable of solving the prompted task. An updated list of all test-cases is available online:

3 Results

Comparing the pass-rates visualized in Figure 2 in detail reveals that the three leading models, gpt-4-turbo-2024-04-09, claude-4-opus-20240229 and gpt-4-1106-preview managed to answer with close pass-rates of $47 \pm 38\%$, $47 \pm 40\%$ and $46 \pm 39\%$, respectively. This indicates that these three models perform similarly well. These numbers correspond to pass@1 counting the success rate from drawn examples. Detailed pass@k rates with k=1, k=5 and k=10 are shown in Figure 3.

Detailed pass rate analysis of the individual test-cases, shown in Figure 4, highlights that most of the test-cases were solvable by at least one LLM. The positive control, return_hello_world, was solved successfully in almost all cases, expressed with a pass-rate of 1.0. Only the codellama model failed in 4 out of 10 runs for this particular test-case.

Details about how often the LLMs used selected Python libraries is shown in Table 1. For example, the skimage library was used in 22 of our human-written reference codes and thus, appears 220 times. The benchmarked LLMs used skimage just in 96 to 132 cases. Also our human-written reference codes were not using opencv, the "cv2" Python package, but the LLMs generated code using "cv2" in 44 to 137 cases.

Common error messages and corresponding counts for each LLM are given in Table 2. Diving deeper into the underlying data reveals for example that gemini-pro often leaves out import statements leading to common error messages such as "name 'np' is not defined".

Sampling the LLMs using our prompts took approximately 20 hours in total. The models gpt-3.5-turbo-1106, both gpt-4 models together, and claude-3-opus-20240229 caused costs of $0.52, $13.02, and $24.58, respectively. All other models did not cause direct costs as their API usage was free.

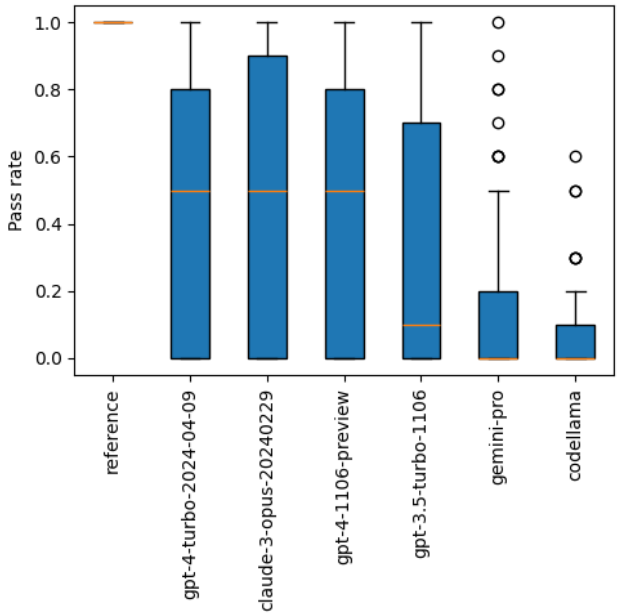

Figure 2. Quantitative pass rate comparison of all tested LLMs and, as a sanity check, the human reference solution: Measured fraction of passed tests visualized as box plot summarizing measurements from 57 test-cases. The corresponding, updated notebook is available online:

4 Discussion

We presented a benchmark for comparing code generation capabilities of LLMs in the Bio-Image Analysis context. Such benchmarks are crucial to decide , e.g. if and how to apply this technology in routine projects, training or advanced applications. It shall be mentioned that we did not use any code-completion tools, such as github co-pilot, to write the test-cases. It is necessary to not use

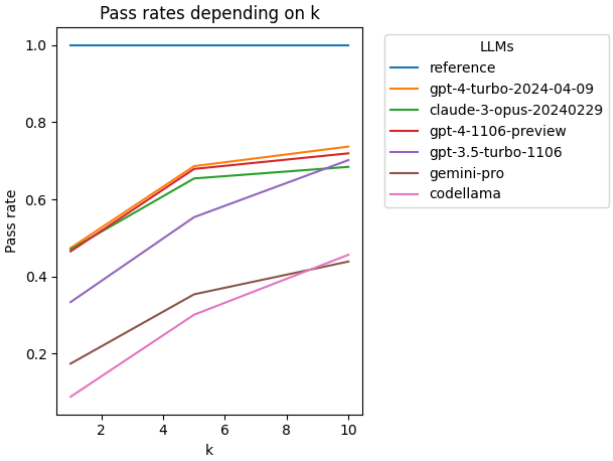

Figure 3. Detailed pass@k with $k = 1$, $k = 5$ and $k = 10$ is a way to estimate the chance to retrieve at least on functional code snippet when generating k samples. The corresponding, updated notebook is available online:

	reference	gpt-4-turbo-2024-04-09	claude-3-opus-20240229	gpt-4-1106-preview	gpt-3.5-turbo-1106	gemini-pro	codellama
numpy	220	434	453	398	360	165	454
scipy	70	123	131	141	76	31	114
skimage	220	129	125	132	115	116	96
cv2	0	63	44	57	144	82	137
pandas	60	100	99	97	90	52	95
pyclesperanto_ prototype	40	0	0	0	0	0	0
vedo	20	0	0	0	0	0	0
umap	20	20	20	20	20	20	20
dask	10	0	0	0	0	0	0
zarr	10	10	10	10	10	10	10
circle_fit	10	0	0	0	1	0	0
statsmodels	0	0	0	0	0	0	2
xarray	0	1	0	0	0	2	2
pillow	0	2	0	0	1	0	0
nibabel	10	17	10	20	11	10	12
SimpleITK	0	2	10	1	8	7	1
trimesh	0	0	0	0	0	1	0
itk	0	2	10	0	9	7	1

Table 1. Used Python libraries in generated code from the tested LLMs. If one generated code snippet contained the same library twice, it is only counted once. The Notebook for generating this table can be found online:

such LLM-based tools while writing the test-cases, because otherwise we might introduce a bias towards underlying LLMs. For example, github-copilot is based on chatGPT. If the test-cases were written using it, the benchmark might misleadingly reveal better performance of chatGPT. Inspecting the detailed reports for which test-cases were not solved by any LLM also revealed a couple of cases, which are considered simple, but LLMs did not solve them. Examples are deconvolve_image, extract_surface_measure_area and open_image_read_voxel_size. Adding new test-cases and modifying such yet-failing test-cases, after a first benchmark has been executed, must be done carefully. A peer-review scheme, e.g. using Github

	gpt-4-turbo-2024-04-09	claude-3-opus-20240229	gpt-4-1106-preview	gpt-3.5-turbo-1106	gemini-pro	codellama
has no attribute	33	45	48	37	43	59
invalid syntax	0	0	1	4	0	58
is not defined	4	5	8	11	203	32
Can't convert object	1	1	3	9	3	13
cannot import	3	5	2	2	6	17
out of range	0	3	0	0	0	4
unexpected keyword argument	15	5	8	7	1	4

Table 2. Snippets of common error messages and how often these were observed when evaluating the generated code from tested models. The Notebook for generating this table can be found online:

pull-requests, is recommended to make sure good scientific practice is maintained. We provide a pull-request template with a short questionaire to support this effort. Additionally, we provide tools that enable to detect if the LLMs are attempting to use Python libraries which are not installed yet. These can be added and the evaluation step can be repeated. As we encourage to only add test-cases that can be implemented with common Python libraries, the manual effort for this quality assurance step is limited. When maintaining this benchmark mid-/long-term modifications should be done with care, to not give advantages to specific LLMs. We limited sample generation for the benchmarking to 10 samples per LLM per test-case. pass@k analysis was done using k=1, k=5 and k=10. The established standard set by [5] is 200 samples and k=1, k=10 and k=100. As our benchmark is in early development (you are reading a preprint), we considered drawing 200 samples as not well-invested compute time and unnecessary costs. Once the benchmark contains more test-cases and models, this decision may be revised. The estimated costs also demonstrate the potential of the technology. Requested code is commonly served within seconds, and drawing hundreds samples from the paid-per-prompt models causes minimal costs depending on the used model. When using LLMs on a daily basis, it appears unreasonable to draw hundreds of samples. Hence, using LLMs for BIA code generation is affordable and cost-efficient. Our benchmark is a single-shot benchmark presenting the prompt to the LLM with no history of a former conversation. In daily use, one can interact with LLMs using chat-interfaces and iteratively engineer a prompt. Thus, the tested LLMs may be more capable than measured in our experiment, when used in a chat scenario. The test-case selection may introduce a certain bias of the sub-discipline we work in. We mostly work with fluorescence microscopy imaging data, often showing nuclei, cytoplasm and membranes. Our test-cases are derived from practical situations we come across often. Mid-/long-term we hope that community contributions to the benchmark's Github repository will allow us to reflect the field more broadly. For example, algorithms processing histological, hyperspectral, or super-resolution imaging data, would be welcome in our test-case collection. On the other hand, we consider test-cases without practical relevance in bio-image analysis should not be part of the benchmark. For example, an algorithm implementing image-reconstruction techniques for computed tomography, or image-classification algorithms for natural images, e.g. showing cats and dogs, are considered off-topic and should not be included. Also intentionally, we included test-cases and prompts which we presume are currently not solvable by LLMs. With this, the

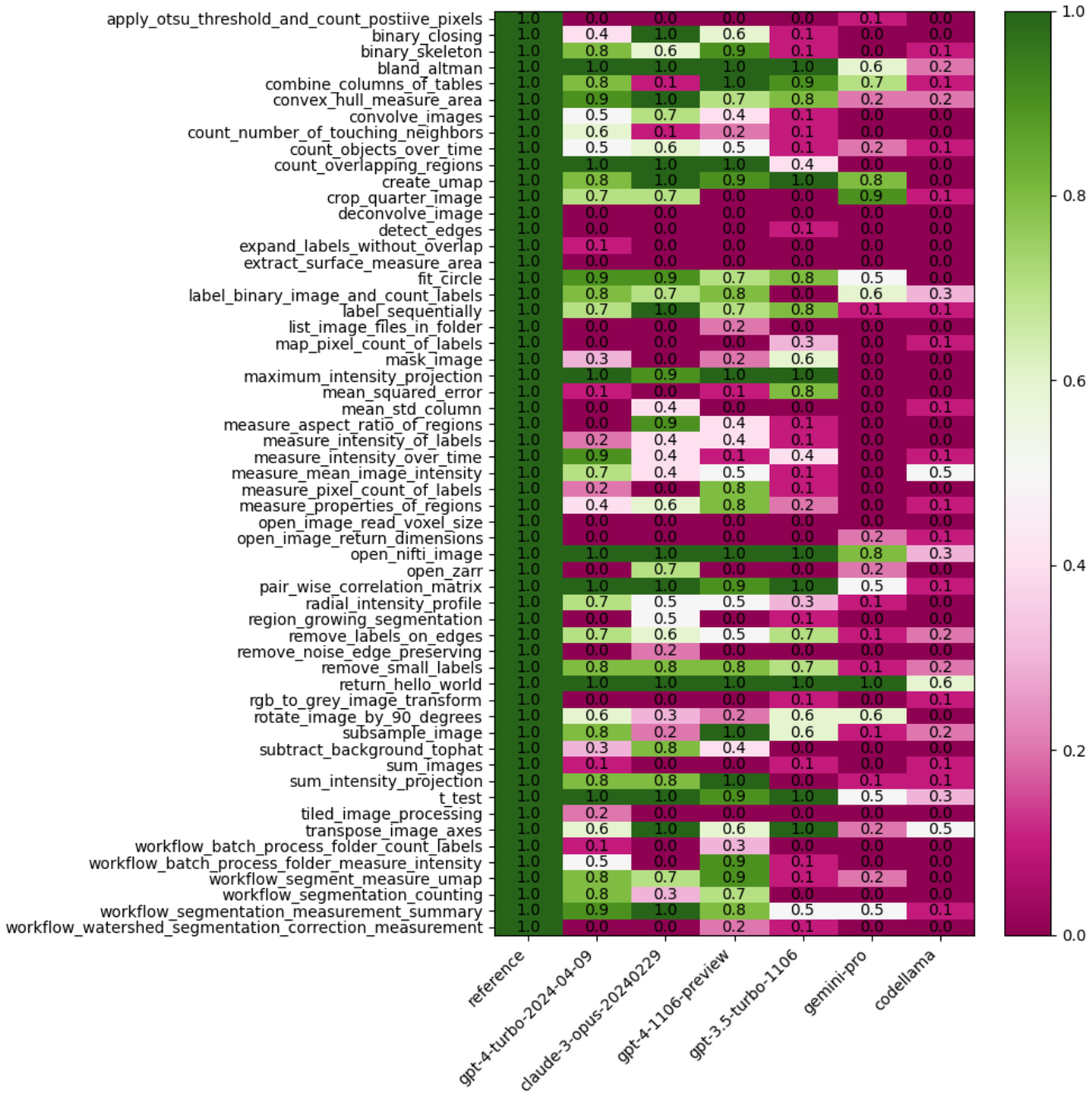

Figure 4. Test-cases and corresponding pass@1 for each LLM. Pass@1 reports the probability that a generated solution works if a user asks the LLM just a single time. The corresponding, updated notebook is available online:

benchmark could guide LLM developers in this field towards more advanced code-generation. In our evaluation of LLMs we see two groups of models: one group seems more capable than the other, expressed by about twice as high pass rates. There might be multiple reasons for this: 1) The tested open-source models are much smaller than the tested commercial models, e.g. codellama has 7B parameters and the GPT models are about two orders of magnitude larger. Obviously, model size limits LLM capabilities. 2) In bio-image analysis, we use some specific Python libraries, such as aicsimageio [3], vedo [14] or pyclesperanto_prototype [8], which might not be mentioned in the training data of some models. On the other hand, in natural image processing other libraries such as OpenCV [11] are commonly used, where we use scikit-image [20] for similar purposes. As natural image processing is also a vivid research field, the LLM's training data may contain more use-cases from that field. Also the DS-1000 benchmark [13], focusing on general data-science use-cases, does not cover scikit-image or opencv. Hence, again the test-cases in our benchmark may enable the LLM community to develop models covering our BIA use-cases and the libraries established in our field better. We aim at developing this benchmark further, together with the

support by the BIA community, by adding more test-cases. With the development of new models, we also would like to adapt the benchmark depending on how LLMs develop. For example, currently upcoming vision-models, LLMs that can take images as input, need to be considered for benchmarking too. Our presented benchmark does not have the capability to test vision-models. Another angle of further development could be efficiency of generated code as proposed earlier [6]. In particular in the context of processing 3D+t imaging data, accelerated image processing techniques can by key [7]. More generally, it might be interesting to investigate different strategies for knowledge representation. For example, fine-tuned models, models with virtually unlimited context length, and retrieval augmented generation are three techniques for storing and accessing information how to use Python libraries. It appears reasonable to compare these techniques quantiatively in the bio-image analysis context.

5 Conclusion

We developed a benchmark for measuring LLM performance in generating code for solving bio-image analysis tasks. It can guide end-users to decide which LLM to use and potentially to pay for when developing bio-image analysis scripts and tools. We also consider this benchmark for LLM-developers in our domain as a metric to guide further development. Last but not least: We encourage the community to send pull-requests with new test-cases to the above-mentioned github repository to ensure the benchmark is covering needs in our field widely. This way, we wish to establish a community-driven approach to benchmarking LLMs for BIA.

References

[1] Anthropic. Introducing the next generation of claude. https://www.anthropic.com/news/claude-3-family, 2024.

[2] J. Austin, A. Odena, M. I. Nye, M. Bosma, H. Michalewski, et al. Program synthesis with large language models. *CoRR*, abs/2108.07732, 2021. URL https://arxiv.org/abs/2108.07732.

[3] E. M. Brown, D. Toloudis, J. Sherman, M. Swain-Bowden, T. Lambert, and AICSImageIO Contributors. Aicsimageio: Image reading, metadata conversion, and image writing for microscopy images in pure python, 2021. URL https://github.com/AllenCellModeling/aicsimageio.

[4] T. B. Brown, B. Mann, N. Ryder, M. Subbiah, J. Kaplan, et al. Language models are few-shot learners. *CoRR*, abs/2005.14165, 2020. URL https://arxiv.org/abs/2005.14165.

[5] M. Chen, J. Tworek, H. Jun, Q. Yuan, H. P. de Oliveira Pinto, et al. Evaluating large language models trained on code. *CoRR*, abs/2107.03374, 2021. URL https://arxiv.org/abs/2107.03374.

[6] M. Du, A. T. Luu, B. Ji, and S.-K. Ng. Mercury: An efficiency benchmark for llm code synthesis, 2024.

[7] R. Haase, L. A. Royer, P. Steinbach, D. Schmidt, A. Dibrov, et al. Clij: Gpu-accelerated image processing for everyone. *Nature Methods*, 17 (1):5–6, 2020. ISSN 1548-7105. doi: 10.1038/s41592-019-0650-1. URL https://doi.org/10.1038/s41592-019-0650-1.

[8] R. Haase, P. Rajasekhar, T. Lambert, grahamross123, J. Nunez-Iglesias, et al. clesperanto/pyclesperanto_prototype: 0.24.2, Dec. 2023. URL https://doi.org/10.5281/zenodo.10432619.

[9] D. Hendrycks, S. Basart, S. Kadavath, M. Mazeika, and A. Arora. Measuring coding challenge competence with apps, 2021.

[10] G. Inc. Introduction to the vertex ai sdk for python. https://cloud.google.com/vertex-ai/docs/python-sdk/use-vertex-ai-python-sdk, 2024.

[11] Itseez. Open source computer vision library. https://github.com/itseez/opencv, 2015.

[12] A. Q. Jiang, A. Sablayrolles, A. Mensch, C. Bamford, D. S. Chaplot, et al. Mistral 7b, 2023.

[13] Y. Lai, C. Li, Y. Wang, T. Zhang, R. Zhong, L. Zettlemoyer, S. W. tau Yih, D. Fried, S. Wang, and T. Yu. Ds-1000: A natural and reliable benchmark for data science code generation, 2022.

[14] M. Musy et al. vedo, a python module for scientific analysis and visualization of 3d objects and point clouds. URL https://doi.org/10.5281/zenodo.2561401.

[15] Ollama. Windows preview. https://ollama.com/blog/windows-preview, 2024.

[16] L. A. Royer. The future of bioimage analysis: a dialog between mind and machine. *Nature Methods*, 20(7):951–952, 2023. ISSN 1548-7105. doi: 10.1038/s41592-023-01930-y. URL https://doi.org/10.1038/s41592-023-01930-y.

[17] L. A. Royer. Omega – harnessing the power of large language models for bioimage analysis. https://doi.org/10.5281/zenodo.8240289, 2023.

[18] B. Rozière, J. Gehring, F. Gloeckle, S. Sootla, I. Gat, et al. Code llama: Open foundation models for code, 2024.

[19] G. Team, R. Anil, S. Borgeaud, J.-B. Alayrac, J. Yu, R. Soricut, et al. Gemini: A family of highly capable multimodal models, 2024.

[20] S. van der Walt, J. L. Schönberger, J. Nunez-Iglesias, F. Boulogne, J. D. Warner, N. Yager, E. Gouillart, T. Yu, and the scikit-image contributors. scikit-image: image processing in python. *PeerJ*, 2:e453, jun 2014. ISSN 2167-8359. doi: 10.7717/peerj.453. URL https://doi.org/10.7717/peerj.453.

[21] A. Yadav and M. Singh. Pythonsaga: Redefining the benchmark to evaluate code generating llm, 2024.