Omega – Harnessing the Power of Large Language Models for Bioimage Analysis

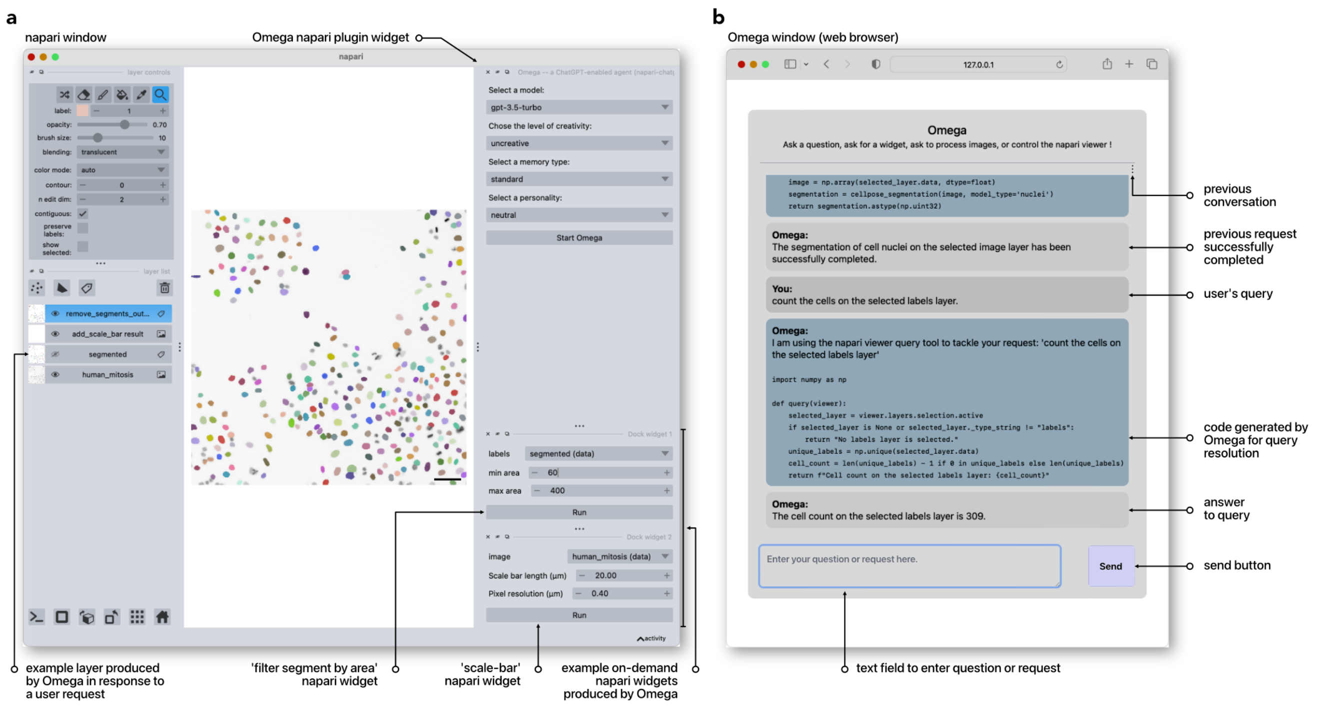
Loïc A. Royer[1,*]

[1]Chan Zuckerberg Biohub, San Francisco, USA.
*Correspondence: loic.royer@czbiohub.org

We stand at a pivotal juncture for Artificial Intelligence. Large Language Models (LLMs) such as ChatGPT[1–3] can now engage in insightful conversations, demonstrating an impressive command of human knowledge and logic. These large language models (LLM) are adept conversationalists and possess an increasingly accurate understanding of numerous scientific and engineering disciplines[4]. ChatGPT, a groundbreaking model from OpenAI, can not only code in various programming languages but also explain and rectify code[5]. This capability is ushering in an era where users, irrespective of their programming skills, can instruct computers to perform complex tasks that would have previously required coding. Natural language is emerging as the hot new programming language. Could we harness these advancements to make image processing and analysis faster, more accessible, and tailor-made to the user's task?

We introduce Omega, an LLM-based conversational agent capable of performing image processing tasks, analyzing images to gather insights, correcting its own coding mistakes, and conducting follow-up quantifications and analyses. For instance, a user can instruct Omega to "segment cell nuclei in the selected image on the napari viewer," then "count the number of segmented nuclei," and finally "return a table that lists the nuclei, their positions and areas" (see Fig. 1, and Supp Video 1 and 2). Moreover, Omega can provide advice and instructions on various image processing and analysis topics. A user can ask Omega to create a "step-by-step plan to segment nuclei in an image," Omega will generate a detailed strategy (see Supp. Video 3). The user can then interactively apply these steps, make changes in response to the outcomes, and ask follow-up questions to complete the task (see also Supp. Video 3). Omega can also create on-demand user interface widgets from user prompts. For example, a user may ask for a "widget that removes segments in a labels layer outside of a range of segment areas" (see Fig 1 and Supp. Video 4) or for a "widget that color projects a 3D image stack so that the hue is proportional to the depth of voxel of max intensity" (See Fig 1, and Supp. Video 5). On-demand widget generation for user-tailored tasks such as specialized image filtering, transformations, visualizations, and more (see Supp. Table 1 for examples) is one of the most valuable aspects of Omega. To facilitate the reuse, modification, and sharing of generated widgets and code snippets, Omega also includes an AI-augmented code editor that keeps all working code generated by Omega, allowing users to rerun the code, modify and improve it using AI-augmented tools such as automatic code commenting, cleanup, fixing, modification, and safety checks (Supp. Fig. 2 and Supp. Video 6). Moreover, the editor also lets users on the same local network easily share code (see Supp. Video 7).

Omega is written in Python as a plugin to napari[6] and leverages the LangChain Python library[7] and OpenAI's application programming interface (API). While Omega works best with OpenAI's ChatGPT, it can also leverage other LLMs, such as Anthropic's Claude models (see Supp. Video 8), and other open-source models via Ollama. Omega is a conversational agent that can converse with the user, like ChatGPT's popular web interface. We utilize the ReAct framework[8] to enable multi-step reasoning and task-specific actions, including access to online sources of information and specialized tools for executing code and interfacing with napari. Omega can also correct its own coding mistakes by receiving feedback on syntax and execution errors encountered (Supp. Video 9).


a
napari window
Omega napari plugin widget
napari
layer controls
layer list
Omega -- a ChatGPT-enabled agent (napari-chat
Dock widget 1
Dock widget 2
activity
example layer produced by Omega in response to a user request
'filter segment by area' napari widget
'scale-bar' napari widget
example on-demand napari widgets produced by Omega
b
Omega window (web browser)
127.0.0.1
previous conversation
previous request successfully completed
user's query
code generated by Omega for query resolution
answer to query
send button
text field to enter question or request

label: 1
opacity: 0.70
brush size: 10
blending: translucent
color mode: auto
contour: 0
n edit dim: 2
contiguous: ✓
preserve labels:
show selected:

remove_segments_out...
add_scale_bar result
segmented
human_mitosis

Select a model: gpt-3.5-turbo
Chose the level of creativity: uncreative
Select a memory type: standard
Select a personality: neutral
Start Omega

labels: segmented (data)
min area: 60
max area: 400
Run

image: human_mitosis (data)
Scale bar length (µm): 20.00
Pixel resolution (µm): 0.40
Run

Omega
Ask a question, ask for a widget, ask to process images, or control the napari viewer !

```
    image = np.array(selected_layer.data, dtype=float)
    segmentation = cellpose_segmentation(image, model_type='nuclei')
    return segmentation.astype(np.uint32)
```

Omega:
The segmentation of cell nuclei on the selected image layer has been successfully completed.

You:
count the cells on the selected labels layer.

Omega:
I am using the napari viewer query tool to tackle your request: 'count the cells on the selected labels layer'

```
import numpy as np

def query(viewer):
    selected_layer = viewer.layers.selection.active
    if selected_layer is None or selected_layer._type_string != "labels":
        return "No labels layer is selected."
    unique_labels = np.unique(selected_layer.data)
    cell_count = len(unique_labels) - 1 if 0 in unique_labels else len(unique_labels)
    return f"Cell count on the selected labels layer: {cell_count}"
```

Omega:
The cell count on the selected labels layer is 309.

Enter your question or request here.

Send

Figure 1. Harnessing the Power of Large Language Models for Bioimage Analysis with Omega. **(a)** Omega is a napari plugin that first appears as a widget that lets users configure and start Omega. Images and other layers (labels, points, shapes) are listed in napari's layer list and accessible to Omega. Omega can add to the layer's list any layer resulting from processing or analysis. Users can ask Omega to make tailor-made widgets that are added to napari. These widgets can input any set of layers and return new layers. **(b)** Upon starting, Omega opens a browser window that displays a chat box page. Users can then begin dialoguing with Omega, asking questions about image processing and analysis, opening images in the napari viewer, asking for a widget, and processing and analyzing images or any other layer supported by napari, such as labels, points, or shapes.

Omega's tools allow it to download files from the web (Supp. Video 10), perform web searches (Supp. Video 11), execute arbitrary Python code (Supp. Video 12), control and query the state as well as contents of the napari viewer (Supp. Video 13), make napari widgets (Supp. Videos 4-5), and query the parameters and documentation of Python functions (Supp. Video 14). Additional tools incorporated into Omega include special-purpose tools that give access to two popular cell and nuclei segmentation algorithms: *cellpose*[9] (Supp. Video 15) and *StarDist*[10] (Supp. Video 1 and 4), as well as to our image denoising software *Aydin*[11] (Supp. Video 16). Omega inherits ChatGPT's Python coding abilities and knowledge (Supp. Video 17). To our surprise, the two best LLMs tested, ChatGPT and Claude, have extensive knowledge of napari's programming interface (Supp. Video 13) as well as other standard and popular Python libraries such as NumPy[12] (Supp. Video 17), scikit-image[13] (Supp. Video 18), and OpenCV[14] (Supp. Video 19). These models can also utilize hardware optimization and acceleration libraries such as *numba*[15] for just-in-time compilation (Supp. Video 20) and CuPy[16] for GPU acceleration (Supp. Video 21). Omega can leverage all this knowledge and tools to perform tasks and answer questions.

However, the promise of this approach also requires prudence. It is well known that LLMs sometimes hallucinate facts and occasionally make trivial reasoning mistakes[4,17]. Indeed, we have observed that Omega sometimes uses functions or parameters that do not exist in the installed libraries (see Supp. Video 9). This is cause for caution because non-expert users might be led astray by an overly confident agent. Moreover, it is incumbent upon the user to explain the task clearly and unambiguously in natural language (note the ambiguity between max and average projection in Supp. Video 20). We are still in the early days of this technology, and rapid progress will hopefully reduce the risks and improve the quality of the reasoning and code produced[18]. In the meantime, Omega implements several features that aim to mitigate these problems, including several introspection routines that check the correctness of generated code by looking for function calls to the wrong library versions, missing import statements, or missing libraries. To understand Omega's reliability, we conducted a reproducibility analysis in which we ran five different complex prompts ten times. In 90% of cases, Omega obtained the correct result (see Supp. Table 2). It often suffices to ask Omega to "try again" to resolve most issues.

This work suggests that LLM-based agents could assist many users in image processing, analysis, and visualization. Beyond just completing tasks, Omega offers an interactive platform that can assist in educating users. Users can ask questions about a particular course of action, why a specific function was used, or for an explanation of some of the concepts used or mentioned by Omega (Supp. Video 4 & 18). Moreover, Omega can operate in a didactic mode, explaining different approaches and presenting choices to the users before acting. Finally, the multilingual capabilities of ChatGPT and other LLMs mean that Omega is accessible to non-English speakers (Supp. Video 22).

Looking ahead, multimodal foundation models have the growing potential to process and understand diverse types of data. Already, state-of-the-art models can incorporate and analyze sound, speech, and videos[19,20]. Omega has access to a Vision tool built upon the vision capabilities of OpenAI's ChatGPT 4. This tool lets Omega interpret the napari viewer's contents visually (Supp. Video 23). This makes it possible for Omega to decide on the best image processing or analysis approach, for example, when segmenting biological objects (see Supp. Video 24). Eventually, these multimodal models will be able to understand extensive multi-dimensional data directly, circumventing the need to write code to process or analyze complex tabular or graph-based data. This evolution will be a step towards more comprehensive and versatile intelligent agents capable of handling more complex tasks across different data modalities, expanding the scope of applications and possibilities in image processing and beyond.

The source code and instructions for using Omega are available at github.com/royerlab/napari-chatgpt.

Acknowledgments:

Thanks to OpenAI for early API access to their ChatGPT 4 models, facilitated by Logan Kilpatrick via Mark Andrew Kittisopikul; Anthropic for early API access to their latest Claude models, facilitated by Josh Batson; and Sandra Schmid for careful proofreading. Thanks to Robert Haase and Adrian Jacobo for bug fixes, suggestions, and discussion.

Ethics Declaration:

The author declares no conflict of interest.

References:

1. OpenAI. GPT-4 Technical Report. Preprint at http://arxiv.org/abs/2303.08774 (2023).
2. Ouyang, L. *et al.* Training language models to follow instructions with human feedback. Preprint at http://arxiv.org/abs/2203.02155 (2022).
3. Sanderson, K. GPT-4 is here: what scientists think. *Nature* **615**, 773–773 (2023).
4. Laskar, M. T. R. *et al.* A Systematic Study and Comprehensive Evaluation of ChatGPT on Benchmark Datasets. Preprint at http://arxiv.org/abs/2305.18486 (2023).
5. Bubeck, S. *et al.* Sparks of Artificial General Intelligence: Early experiments with GPT-4. Preprint at http://arxiv.org/abs/2303.12712 (2023).
6. Sofroniew, N. *et al.* napari: a multi-dimensional image viewer for Python. *Zenodo* (2022).
7. Chase, H. LangChain, 10 2022. *URL Httpsgithub Comhwchase17langchain*.
8. Yao, S. *et al.* ReAct: Synergizing Reasoning and Acting in Language Models. Preprint at http://arxiv.org/abs/2210.03629 (2023).
9. Pachitariu, M. & Stringer, C. Cellpose 2.0: how to train your own model. *Nat. Methods* **19**, 1634–1641 (2022).
10. Weigert, M., Schmidt, U., Haase, R., Sugawara, K. & Myers, G. Star-convex Polyhedra for 3D Object Detection and Segmentation in Microscopy. in *2020 IEEE Winter Conference on Applications of Computer Vision (WACV)* 3655–3662 (IEEE, 2020). doi:10.1109/WACV45572.2020.9093435.

11. Solak, A. C., Loic A. Royer, Abdur-Rahmaan Janhangeer & Kobayashi, H. royerlab/aydin: v0.1.15. (2022) doi:10.5281/ZENODO.5654826.
12. Harris, C. R. *et al.* Array programming with NumPy. *Nature* **585**, 357–362 (2020).
13. Van der Walt, S. *et al.* scikit-image: image processing in Python. *PeerJ* **2**, e453 (2014).
14. Bradski, G. The openCV library. *Dr Dobbs J. Softw. Tools Prof. Program.* **25**, 120–123 (2000).
15. Lam, S. K., Pitrou, A. & Seibert, S. Numba: A llvm-based python jit compiler. in *Proceedings of the Second Workshop on the LLVM Compiler Infrastructure in HPC* 1–6 (2015).
16. Nishino, R. & Loomis, S. H. C. Cupy: A numpy-compatible library for nvidia gpu calculations. *31st Confernce Neural Inf. Process. Syst.* **151**, (2017).
17. Li, J., Cheng, X., Zhao, W. X., Nie, J.-Y. & Wen, J.-R. HaluEval: A Large-Scale Hallucination Evaluation Benchmark for Large Language Models. Preprint at http://arxiv.org/abs/2305.11747 (2023).
18. Peng, B. *et al.* Check your facts and try again: Improving large language models with external knowledge and automated feedback. *ArXiv Prepr. ArXiv230212813* (2023).
19. Wu, C. *et al.* Visual ChatGPT: Talking, Drawing and Editing with Visual Foundation Models. (2023) doi:10.48550/ARXIV.2303.04671.
20. Royer, L. A. The future of bioimage analysis: a dialog between mind and machine. *Nat. Methods* **20**, 951–952 (2023).
21. Kirillov, A. *et al.* Segment Anything. Preprint at http://arxiv.org/abs/2304.02643 (2023).
22. Kojima, T., Gu, S. S., Reid, M., Matsuo, Y. & Iwasawa, Y. Large Language Models are Zero-Shot Reasoners. Preprint at http://arxiv.org/abs/2205.11916 (2023).

Methods:

In the following, we provide details on Omega's different components and how they are implemented, from the chat window web server and the code editor to the cascaded LLM approach, different tools, and Python code repair strategies. The most up-to-date version of Omega's code can be consulted at:
https://github.com/royerlab/napari-chatgpt.
The version corresponding to this manuscript is at the branch named manuscript_03_2024.

User interface and usage. Omega is provided as a napari plugin following the latest plugin standard (npe2). The plugin was built by following the instructions described here. Omega's 'widget plugin' provides a simple interface that lets users configure different options, start Omega, or open the code editor (see Fig 1a). Once the user starts Omega (see start button on Supp. Fig. 1), the plugin starts a web server at the local address 127.0.0.1 that hosts the Omega Chat page. The plugin then opens a web browser window at that address to display the page. At that point, the user can start dialoguing with Omega. Ideally, this chat window is side-by-side with the napari window so that the user can see both the conversation displayed on the browser and the outcome shown in the viewer.

Code editor. Omega saves all working code in a 'code editor' that supports code highlighting and completion. It has AI-augmented tools for automatic code commenting, cleanup, fixing, modification, and safety check (see Supp. Fig. 2). Users can keep all snippets of code and widgets for reuse from one napari session to another, make any manual changes or fixes necessary, and rerun these. Moreover, we implemented a code-sharing feature by which any running instance of Omega's code editor can find other instances in the local network (via multicast) and share code snippets with others. Notably, the code editor is independent of Omega's agent and only requires access to any LLM APIs to access its AI-augmented features. If no LLM API keys are provided, the code editor still runs but without any AI features.

Overall Architecture. Supp. Fig. 3 shows an overview of Omega's architecture, showing the main UI elements at the top: napari itself, the chat window, and the AI-augmented code editor. It also shows the main backend elements: Omega's ReAct agent, the web server, the editor's AI tools, and the Python code repair module. The figure also shows Omega's cascaded architecture, with a main dialog loop driven by a *main LLM* and a series of tools that internally use a delegated *tool LLM* that leverages online resources, the napari viewer itself, and 3rd party libraries such as Cellpose and StarDist.

Jupyter export. Omega can save all conversations in runnable Jupyter notebooks. This is done with the standard Jupyter notebook creation and manipulation library 'nbformat'. In addition, screenshots of the viewer are regularly appended to the notebook to document what was visible on the napari viewer during the conversation.

Omega configuration. The Omega widget (see Supp. Fig. 1) allows users to set: (i) the LLM model (GPT3.5, GPT4, Claude, etc.); (ii) the model's creativity level (normal, slightly creative, moderately creative, creative) that corresponds to different temperature settings of the LLM model. A temperature near zero (normal creativity) means that the model is nearly deterministic in its answers – which is desirable in factual dialoguing and code generation. When the temperature increases, the LLM models explore more atypical and often creative responses – but they also tend to make more factual, reasoning, and coding mistakes. (iii) The type of agent memory used (infinite, bounded, and hybrid). In the case of infinite memory, the agent remembers every word of the conversation, which in practice only works for LLM models with extensive input text lengths such as GPT4 (32k) or Claude (100k). In contrast, the bounded memory only remembers the last k messages exchanged between the agent and user. The hybrid memory precisely remembers the last k messages and summarizes all previous messages. (iv) The agent's personality (neutral, coder, prof, yoda, mobster) modulates the style and tone of the conversation. The following options are related to code generation and the different strategies adopted to mitigate and prevent errors: (v) The option "fix missing imports" controls whether to check the generated code, identify missing imports, and prepend them to the code. (vi) the option "fix bad function calls" controls whether to verify if the function calls in the generated code correspond to functions in the packages installed in the Python environment. (vii) the "Install missing packages" controls whether to list all Python packages required by generated code, compare that list with the list of installed packages, and proceed to install those missing. (viii) The "Autofix coding mistakes" option controls whether Omega will try to fix its own coding mistakes when exceptions occur when interacting with the napari viewer. Similarly, (ix) the "Autofix widget coding mistakes" controls whether Omega will try to fix its own coding mistakes when exceptions occur while making a new widget. Finally, (x) the last option, "High console verbosity," controls Omega's console verbosity level.

Chat server. The chat page is served by uvicorn, an ASGI web server implementation for Python that uses FastAPI to communicate between the chat box and Python. It leverages Jinja2 as the template engine for generating the served HTML page. The chat box and Python communications are handled via a web socket on the client side and a FastAPI endpoint on the server side. Messages between the agent and user are exchanged as JSON-encoded dictionaries.

Omega ReAct Agent. Omega is implemented as LangChain's *ConversationalChatAgent,* which uses the ReAct framework[8] to decide which tool to use and uses memory to remember the previous messages in the conversation. By default, Omega uses a modified version of LangChain's hybrid conversational memory (see code here).

Prompt engineering for Python code generation. The building of Omega required much effort in "Prompt Engineering," which is the art of designing prompts that nudge LLMs into producing the correct answers expected by users. LLMs are known to require very explicit – if not obvious – instructions. For instance, simply adding to the prompt: "Let's think step by step," improves the quality of results[22]. In the case of Omega, we had to make explicit that: "You are an expert Python programmer with deep expertise in image processing and analysis," that: "Your responses are accurate and informative," that it should "Make sure that the code is correct, complete and functional without any missing code, data, or calculations". LLM prompts used for code generation also contain the list of layers in the napari viewer, the current Python version number, and the names and versions of all image-processing relevant libraries installed in the environment. Our experiments show that ChatGPT knows about differences in the parameters of a function across different package versions. This means explicitly providing the information about which specific library versions are installed is critical for correct code generation. We also had to provide detailed instructions so that the code generated could be easily interfaced with Omega's code, thus facilitating interaction with the napari viewer. A simple strategy is to ask the LLM to produce a function with a well-defined signature (input parameters, their types, and return type) and load the code dynamically as a Python module.

Omega's Tools. Omega has at its disposal several tools that give it the ability to (i) search text and images on the web and Wikipedia, access a Python REPL (Read-Eval-Print Loop) for executing arbitrarily non-napari related code, (ii) gather detailed information about Python functions available in the environment, (iii) get information about the latest exceptions that occurred, (iv) obtain information about the state of the napari viewer and about the layers present, (v) make and add widgets to napari's UI, (vi) use special-purpose libraries for cell segmentation and image denoising, and (vii) *see* the contents of the napari viewer with GPT-4 vision to inform its next steps. Following the ReAct[8] approach, the agent maintains a conversation with the user and can use tools to answer questions or perform tasks. This is achieved by listing the available tools and their description in the prompt sent to the LLM. Part of the dialog related to tool usage is internal to the agent and not shared with the user.

Cascaded LLM Architecture. In Omega, we choose a cascaded LLM approach where, besides the LLM running the ReAct agent, the tools invoke LLMs to generate and introspect code and summarize the text. This avoids polluting the main dialog loop trace with long blocks of generated code and makes it possible to tailor prompts to each code generation task. For example, when the main LLM running the ReAct agent decides to invoke the 'widget maker tool' it calls that tool and forwards the user (or its own) request to the tool. The tool itself is a function that takes the request and inserts it into a long and complex prompt that guides the LLM in generating the Python code for the widget.

Custom protocol for tool communication. Most conversational ReAct-based agents use JSON-formatted dictionaries to allow communication between LLMs and the tools. This works well when simple short text strings are exchanged between the LLM and the tool, but this fails for arbitrary code because of all the complexities entailed, such as escaping reserved characters. Imposing such a high competence bar on the LLM by requiring it to produce a very complex JSON string is unreasonable. To solve this issue and reduce the complexity of the syntax that the LLM has to adhere to, we use a simplified multi-line key-value format (see code here). Recent versions of OpenAI's models have built-in support for 'function calling', eliminating this problem.

Python code introspection and repair. Careful prompt engineering can be highly effective at ensuring that the generated code is synthetically correct, that function calls refer to existing and available functions, and that the code is interfaceable with the rest of Omega's code. However, despite our best efforts, there are cases in which the generated code is incorrect. Omega implements several mitigation strategies that reduce the probability of error.

Adding missing import statements. The first typical type of code generation error that we noticed is missing import statements. To address this, Omega implements a particular routine using the code generation LLM to introspect the code by listing all missing import statements. This might seem paradoxical: why should the LLM make a mistake during code generation but be able to catch it during verification? The explanation is that code generation is more challenging than code verification because it requires both Python and Application domain knowledge. In contrast, code verification only requires knowledge of Python and its libraries – generally, the more restricted and well-defined the task, the better the outcome.

Installing missing packages. Adding missing import statements is only helpful if the corresponding libraries are installed in the current Python environment. Using the same code introspection approach, we ask the code generation LLM instance to list all Python packages required to run the generated code. This list of packages is compared to installed packages, and only missing packages are installed using pip. Edge cases like GPU accelerated libraries like Tensorflow or CuPy are handled with special rules. Omega asks permission to install these missing packages to prevent issues with the Python environment.

Incorrect function call detection and repair. An additional mitigation approach involves enumerating all function calls occurring in the code using standard Python language introspection features and checking that these functions exist and that the corresponding packages are installed. This detection step is highly reliable because it does not use LLMs. Once an incorrect function call is detected, we carefully construct a specialized prompt that combines all the information, particularly the correct function signature, and asks the LLM code generation instance to fix the code accordingly.

Context-aware code repair upon code execution error. Once the above fixes are applied to the generated code, then Omega executes that code. If exceptions are detected during execution, the code and exception(s) are provided to the code generation LLM instance. With a specialized prompt, the LLM is explicitly asked to fix the code, given a detailed error description. This process can be repeated recursively until no exceptions occur or the maximum number of repair steps has been reached. Importantly, Omega has a mechanism that forwards task-specific coding instructions used during code generation to be available during code repair.

Napari integration and communication. Giving Omega access to the napari viewer is not trivial because of the threading model mismatch between the agent controller and the napari viewer. The agent runs in async mode per LangChain's implementation, while napari's threading model is inherited from the Qt cross-platform application framework. We address this using a thread-safe bidirectional asynchronous communication queue that establishes a bridge between Omega and napari (see the `NapariBridge` class here). The queue passes code as a string and then executes that code using napari's thread-worker functionality. All captured standard output strings are captured and returned to Omega. Exceptions are dealt with according to the error mitigation strategies described above. The following details the tools that provide Omega access to napari.

Napari viewer query tool. This tool lets Omega gather any information about the state of the napari viewer or any of the layers (images, labels, points, etc.) currently loaded. This is achieved by carefully crafting a prompt incorporating the user's question, information on layers that are present in the viewer, and task-specific coding instructions. In this prompt, the code generation LLM instance is asked to write a '`query(viewer: Viewer)`' function that takes the viewer as a parameter and prints out the answer to the user's question.

Napari viewer control tool. Similarly, this tool lets Omega control the napari viewer, such as changing the state of the viewer's canvas, adding and removing layers, etc. In that case, we carefully crafted a prompt incorporating the user's question, information on layers present in the viewer, and task-specific coding instructions. In this prompt, the code generation LLM instance is asked to write a script that is then executed.

Widget maker tool. This tool takes the user's plain text description of a widget or instructions on modifying a previously generated widget and adds that widget to napari's user interface. For instance, if the user asks for a 'Gaussian filter with a sigma parameter,' this tool will make the corresponding widget with a single float parameter. This automatic user interface generation is made possible by the MagicGUI library as part of the standard plugin infrastructure of napari. The generated code goes through all the checks and verifications described above.

Napari viewer vision tool. This tool gives a 'vision'. Sense to Omega. Currently, this is only supported using OpenAI's ChatGPT 4 vision capabilities, but this could be extended to support other and upcoming vision models. Omega can use this tool in two ways: (i) look at the whole viewer or (ii) focus on a single layer. In the first case, the tool takes a snapshot of the whole napari viewer window, sends that picture to the vision model, and obtains a textual description of the image. In the second case, the tool hides all layers except the selected one, resets the viewer to fully show the layer, takes a snapshot of just the viewer's canvas, sends that picture to the vision model, and obtains a textual description of the image. Occasionally, the ChatGPT 4 model refuses to describe the snapshot image. This is documented here and probably caused by over-zealous security measures preventing the model from describing people. These cases are detected, and the query is repeated four times until it works.

Cell segmentation and image denoising tools. We can't expect LLMs, even the best ones such as ChatGPT 4 or Anthropic's Claude, to know about the latest version of state-of-the-art bioimage analysis libraries such as Cellpose and StarDist for cell and nuclei segmentation and Aydin (aydin.app) for image denoising. To ensure their availability and facilitate their usage, these libraries are integrated explicitly through specific interfacing functions that expose a subset of relevant parameters from these libraries. Specially crafted prompts explain in detail how to use these functions, how to choose between the different variants, and how to set the parameters.

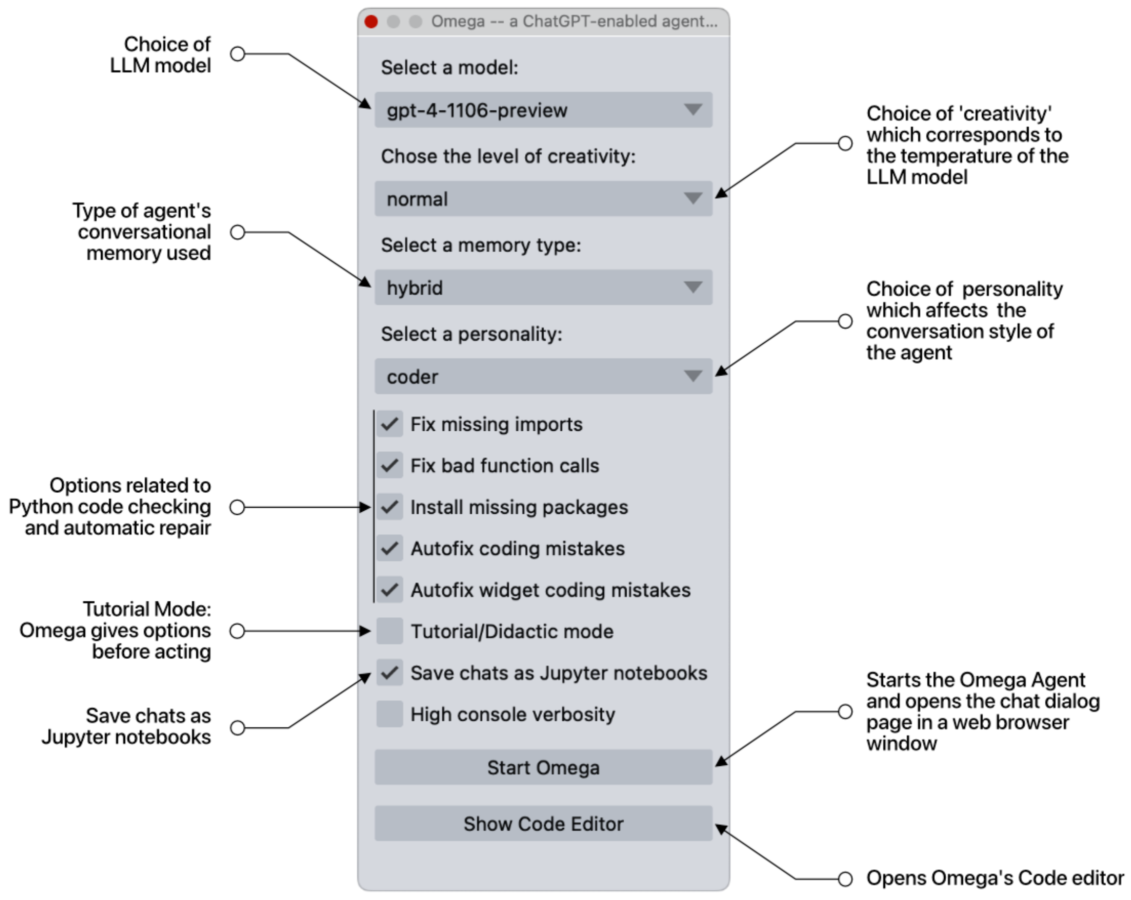
'Classic' cell segmentation. In addition to Cellpose and StarDist, we implemented a straightforward yet configurable threshold-based 'classic' segmentation algorithm using scikit-image functions. This simple algorithm is a reasonably practical baseline for segmenting 3D images of nuclei. Images are first normalized, then using a disk- or ball-based footprint of radius 2; the image is eroded several times. Next, one of the following thresholding functions is applied: otsu, yen, li, minimum, triangle, mean, or isodata. The LLM makes the choice based on user prompt instructions. Next, the closing and opening operators are applied several times to remove potential small segments. An optional routine that uses watersheds to split the under-segmented segments is available. Finally, the resulting binary image is labeled, and a label layer is returned and added to napari.

Didactic posture. To enhance the interactive learning experience within Omega, we introduced a "Didactic mode". This mode can be activated at Omega's startup (see Supp. Fig. 1). In this mode, Omega adopts a didactic posture, initially presenting the user with a summarization of the task followed by a proposal of multiple strategies to address the request. Users can review these methodologies before initiating any processes, allowing for a conscious selection of the preferred approach. This educational orientation extends further; the system is designed to not only guide but also to educate. It expounds upon the rationales underlying various options, delves into image processing and analytical concepts, and elucidates their practical applications, strengths, and weaknesses. This is achieved by adding instructions to that effect in the agent's system prompt. Such a feature turns Omega into not just a tool for task execution but also a tutor for concept learning.

API key vault. The Omega API key vault system is designed to provide a secure method of storing and retrieving API keys required to interface with services such as OpenAI, Anthropic, and others. It ensures that keys are encrypted and accessible only with a proper password, thereby protecting API keys from unauthorized access. The system includes a user-friendly interface that: (i) the first time prompts the user for an API key and password to secure it and (ii) subsequently only asks for the password. Some API keys can be set as an environment variable; this is recognized, thus easing the API key management process, but at the risk of having an exposed unencrypted key on the user's system. The Omega API key vault utilizes the Fernet symmetric encryption standard, ensuring that API keys are stored securely on the user's local system and can only be accessed by providing the correct password. The encryption process is bolstered by a password-based key derivation function (PBKDF2HMAC) and a SHA256 hashing algorithm, which provide a secure method of transforming the password into an encryption key.

Videos. The videos were recorded with OSX's QuickTime player and were sped up by a factor 2x. Supp. Table 3 lists all prompts for all videos.

Supplementary material:


Choice of LLM model
Choice of 'creativity' which corresponds to the temperature of the LLM model
Type of agent's conversational memory used
Choice of personality which affects the conversation style of the agent
Options related to Python code checking and automatic repair
Tutorial Mode: Omega gives options before acting
Save chats as Jupyter notebooks
Starts the Omega Agent and opens the chat dialog page in a web browser window
Opens Omega's Code editor
Omega -- a ChatGPT-enabled agent...
Start Omega
Show Code Editor

Select a model: gpt-4-1106-preview

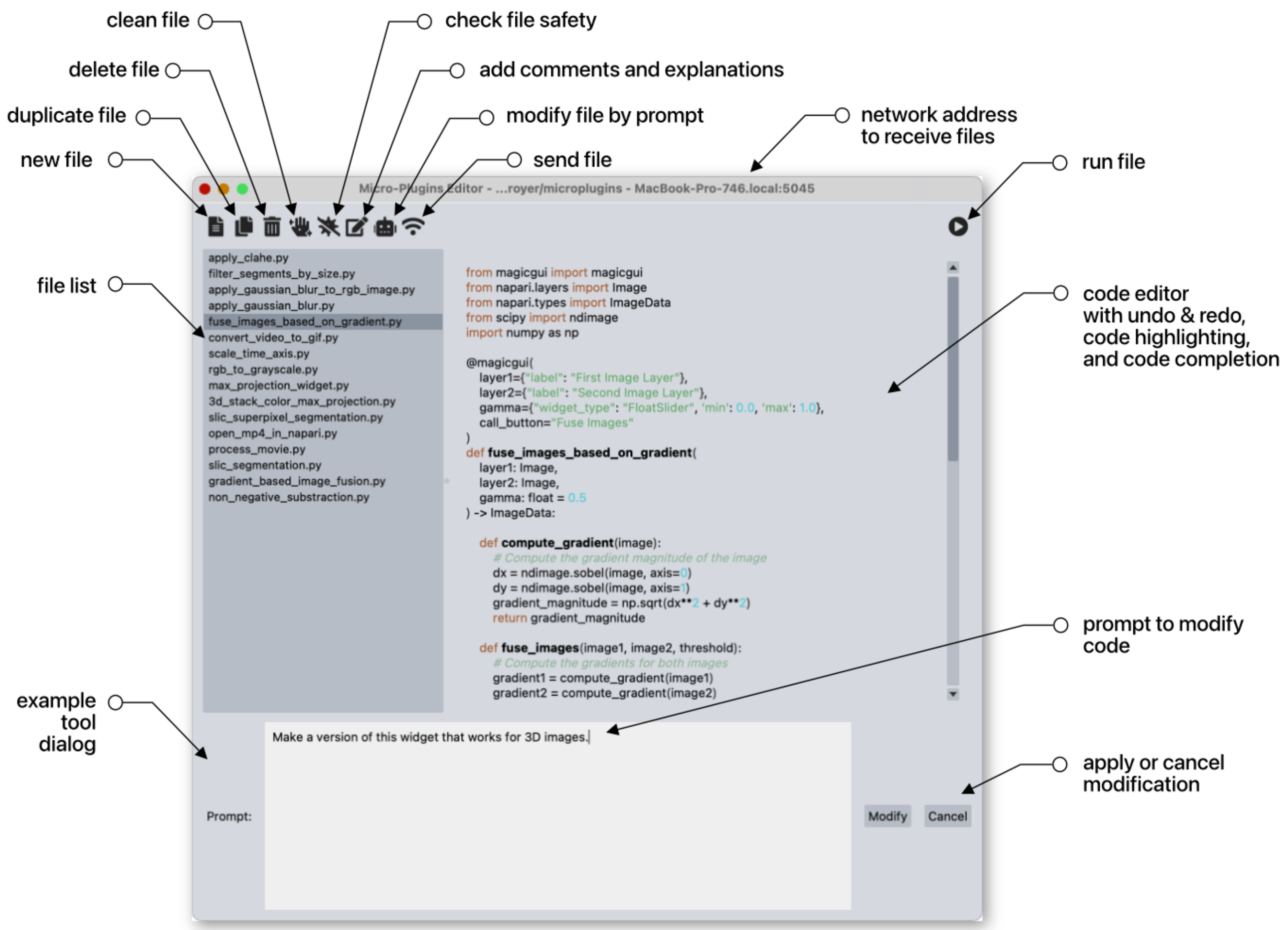
Chose the level of creativity: normal

Select a memory type: hybrid

Select a personality: coder

- [✓] Fix missing imports
- [✓] Fix bad function calls
- [✓] Install missing packages
- [✓] Autofix coding mistakes
- [✓] Autofix widget coding mistakes
- [ ] Tutorial/Didactic mode
- [✓] Save chats as Jupyter notebooks
- [ ] High console verbosity

Supplementary Figure 1. Enhanced Code Editor Interface in Omega: Collaborative and Intelligent Code Management for Image Processing. Users can use Omega's main widget to select different options, including the LLM model's type and version, the level of creativity (which increases the model's temperature), the type of conversational memory used, and the agent's personality. Other parameters relate to code checking and automatic repair. To begin using Omega, click the "Start Omega" button, and a browser window will open, displaying the agent's chat box. To open the code editor, click on "Show Code Editor". Note: the code does not need Omega to be started or LLM API keys to be minimally functional.

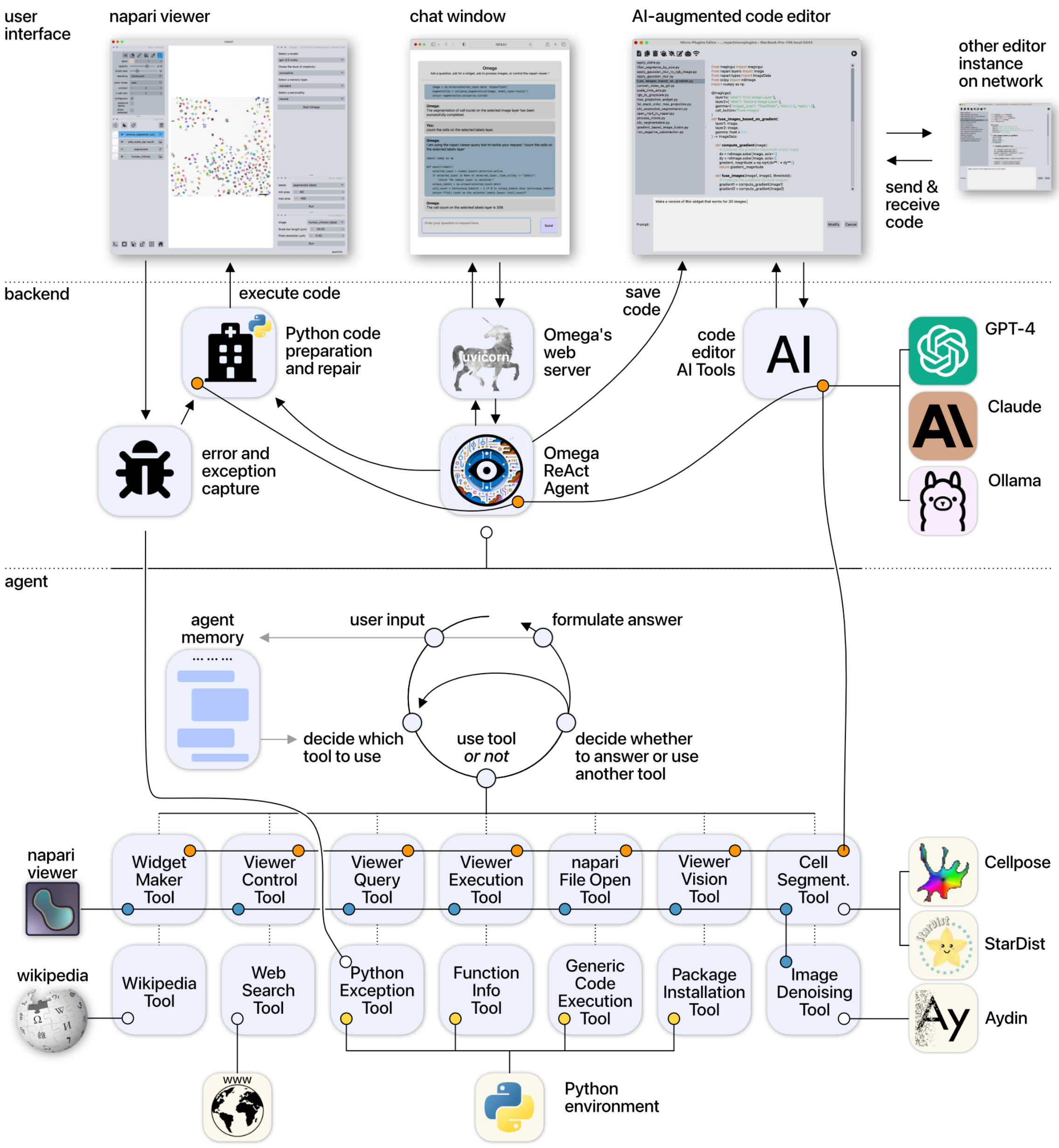

clean file
check file safety
delete file
add comments and explanations
duplicate file
modify file by prompt
network address to receive files
new file
send file
run file
file list
code editor with undo & redo, code highlighting, and code completion
prompt to modify code
example tool dialog
apply or cancel modification
Micro-Plugins Editor - ...royer/microplugins - MacBook-Pro-746.local:5045
Prompt:
Modify
Cancel

```
apply_clahe.py
filter_segments_by_size.py
apply_gaussian_blur_to_rgb_image.py
apply_gaussian_blur.py
fuse_images_based_on_gradient.py
convert_video_to_gif.py
scale_time_axis.py
rgb_to_grayscale.py
max_projection_widget.py
3d_stack_color_max_projection.py
slic_superpixel_segmentation.py
open_mp4_in_napari.py
process_movie.py
slic_segmentation.py
gradient_based_image_fusion.py
non_negative_substraction.py
```

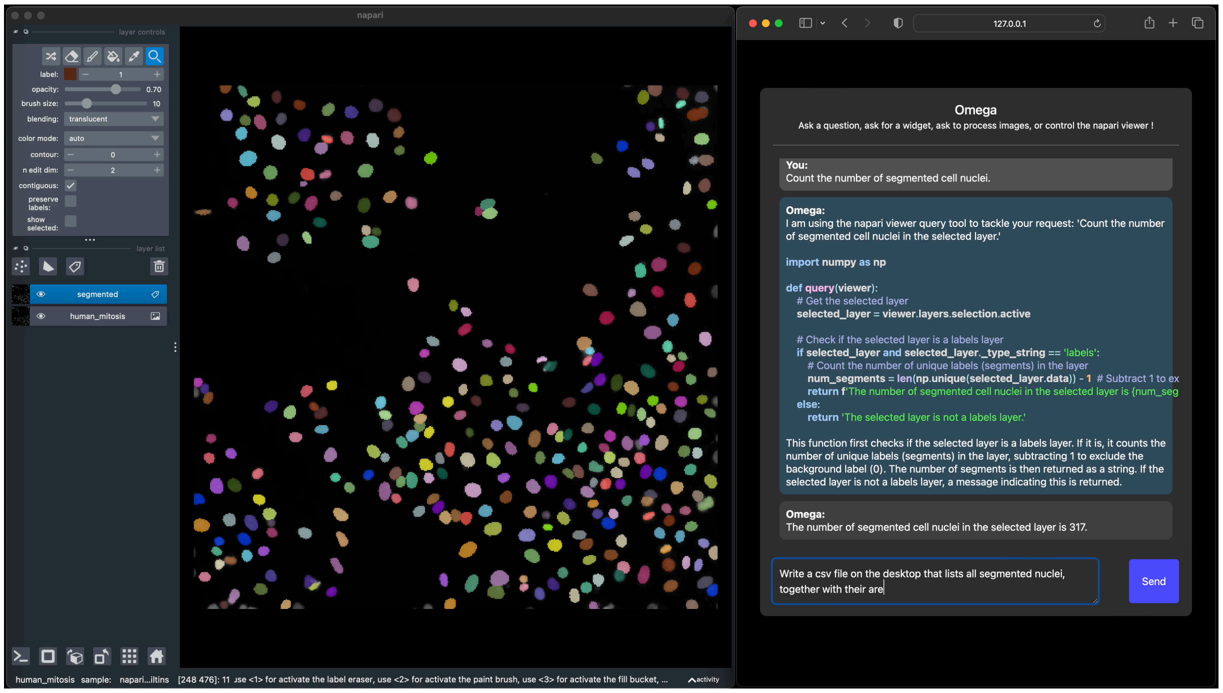
```
from magicgui import magicgui
from napari.layers import Image
from napari.types import ImageData
from scipy import ndimage
import numpy as np

@magicgui(
  layer1={"label": "First Image Layer"},
  layer2={"label": "Second Image Layer"},
  gamma={"widget_type": "FloatSlider", 'min': 0.0, 'max': 1.0},
  call_button="Fuse Images"
)
def fuse_images_based_on_gradient(
  layer1: Image,
  layer2: Image,
  gamma: float = 0.5
) -> ImageData:

  def compute_gradient(image):
    # Compute the gradient magnitude of the image
    dx = ndimage.sobel(image, axis=0)
    dy = ndimage.sobel(image, axis=1)
    gradient_magnitude = np.sqrt(dx**2 + dy**2)
    return gradient_magnitude

  def fuse_images(image1, image2, threshold):
    # Compute the gradients for both images
    gradient1 = compute_gradient(image1)
    gradient2 = compute_gradient(image2)
```

Make a version of this widget that works for 3D images.

Supplementary Figure 2. Omega's AI-augmented code editor. Omega's code editor is a key component for managing and refining code for image processing and analysis tasks. The editor is designed to support code highlighting and completion and is enhanced with LLM-augmented tools for automatic code commenting, cleaning, fixing, modification, and safety checks. This allows users to maintain a library of code snippets and widgets, encouraging code reuse and collaboration through a code sharing feature that enables sending code snippets across local network to other code editor instances.



Supplementary Figure 3. Omega's System Architecture. Diagram illustrating the Omega system architecture, encompassing both the user interface and backend components. The user interface is displayed at the top, consisting of the napari viewer, chat window, and AI-enhanced code editor. Key backend elements include the Omega ReAct agent, web server, AI tools within the editor, and the Python code repair module. The architecture employs a cascaded design, with a central dialog loop orchestrated by a primary language model. This model coordinates a suite of specialized tools, each powered by a secondary tool-specific language model that can access online resources, interact with the napari viewer, and integrate functionalities from third-party libraries such as Cellpose and StarDist.



Omega
Ask a question, ask for a widget, ask to process images, or control the napari viewer !

You:
Count the number of segmented cell nuclei.

Omega:
I am using the napari viewer query tool to tackle your request: 'Count the number of segmented cell nuclei in the selected layer.'

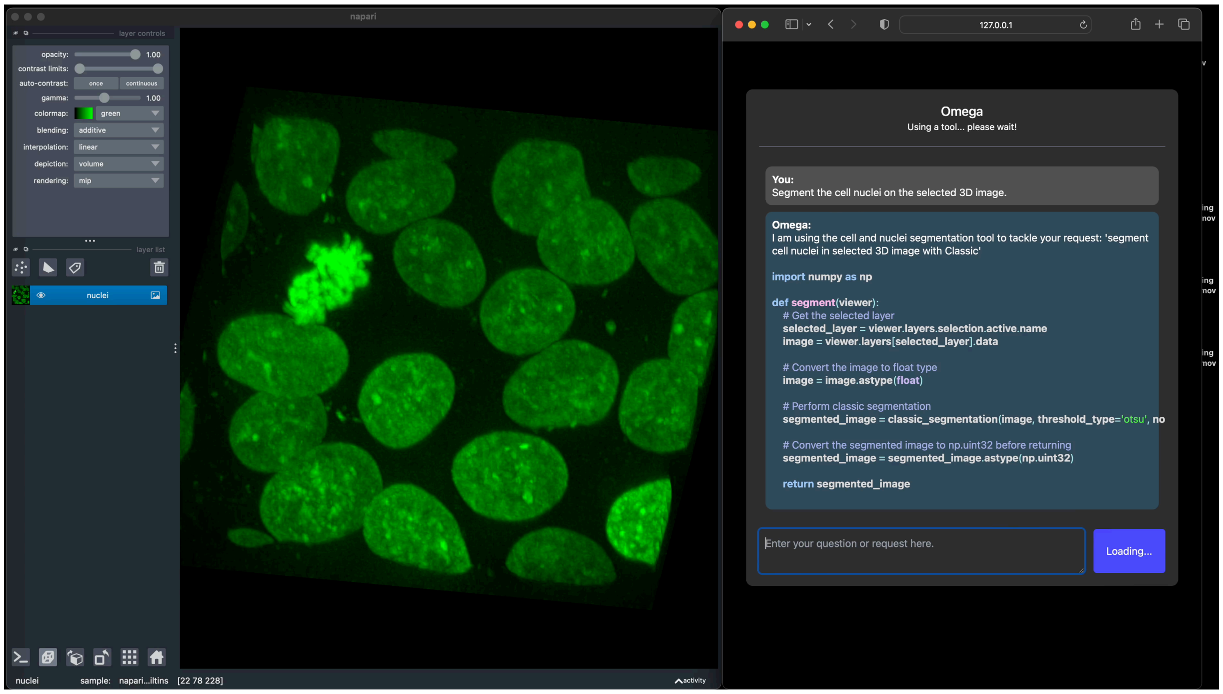
```
import numpy as np

def query(viewer):
    # Get the selected layer
    selected_layer = viewer.layers.selection.active

    # Check if the selected layer is a labels layer
    if selected_layer and selected_layer._type_string == 'labels':
        # Count the number of unique labels (segments) in the layer
        num_segments = len(np.unique(selected_layer.data)) - 1  # Subtract 1 to ex
        return f'The number of segmented cell nuclei in the selected layer is {num_seg
    else:
        return 'The selected layer is not a labels layer.'
```

This function first checks if the selected layer is a labels layer. If it is, it counts the number of unique labels (segments) in the layer, subtracting 1 to exclude the background label (0). The number of segments is then returned as a string. If the selected layer is not a labels layer, a message indicating this is returned.

Omega:
The number of segmented cell nuclei in the selected layer is 317.

Write a csv file on the desktop that lists all segmented nuclei, together with their are

Supplementary Video 1. Omega can segment nuclei with StarDist and perform follow-up analysis. The video showcases Omega's ability to segment cell nuclei in a 2D image using Stardist. Omega successfully segments the nuclei and adds a label layer to the napari viewer. With further instructions, Omega can count the segmented nuclei and create a CSV file on the desktop folder of the machine. This file contains coordinates and areas of all segments, sorted by decreasing area, with one segment per row. Omega also opens the file using the system's default CSV viewer. The video has been sped up by a factor of 2.



Omega
Using a tool... please wait!

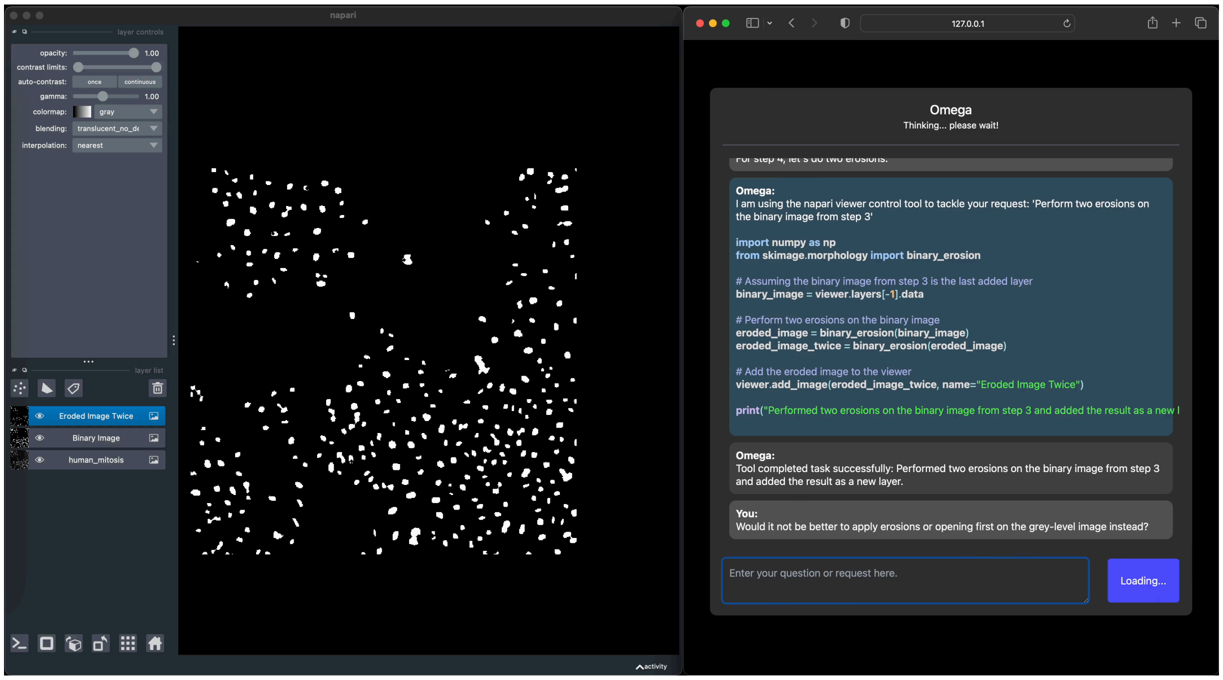
You:
Segment the cell nuclei on the selected 3D image.

Omega:
I am using the cell and nuclei segmentation tool to tackle your request: 'segment cell nuclei in selected 3D image with Classic'

```
import numpy as np

def segment(viewer):
    # Get the selected layer
    selected_layer = viewer.layers.selection.active.name
    image = viewer.layers[selected_layer].data

    # Convert the image to float type
    image = image.astype(float)

    # Perform classic segmentation
    segmented_image = classic_segmentation(image, threshold_type='otsu', no

    # Convert the segmented image to np.uint32 before returning
    segmented_image = segmented_image.astype(np.uint32)

    return segmented_image
```

Enter your question or request here.

Supplementary Video 2. Omega can segment nuclei in a 3D image. This video shows how Omega segments the nuclei in a 3D image displayed in the napari viewer. Omega uses a specialized tool for cell and nuclei segmentation and employs a 'classic' approach that combines single thresholding, specifically Otsu, with watershed splitting to prevent under-segmentation. After segmentation, Omega adds a labels layer to the viewer, and we inquire about the number of segments detected. The response is 27. The video has been sped up by a factor of 2.



opacity: 1.00
contrast limits:
auto-contrast: once continuous
gamma: 1.00
colormap: gray
blending: translucent_no_de
interpolation: nearest

Eroded Image Twice
Binary Image
human_mitosis

For step 4, let's do two erosions.

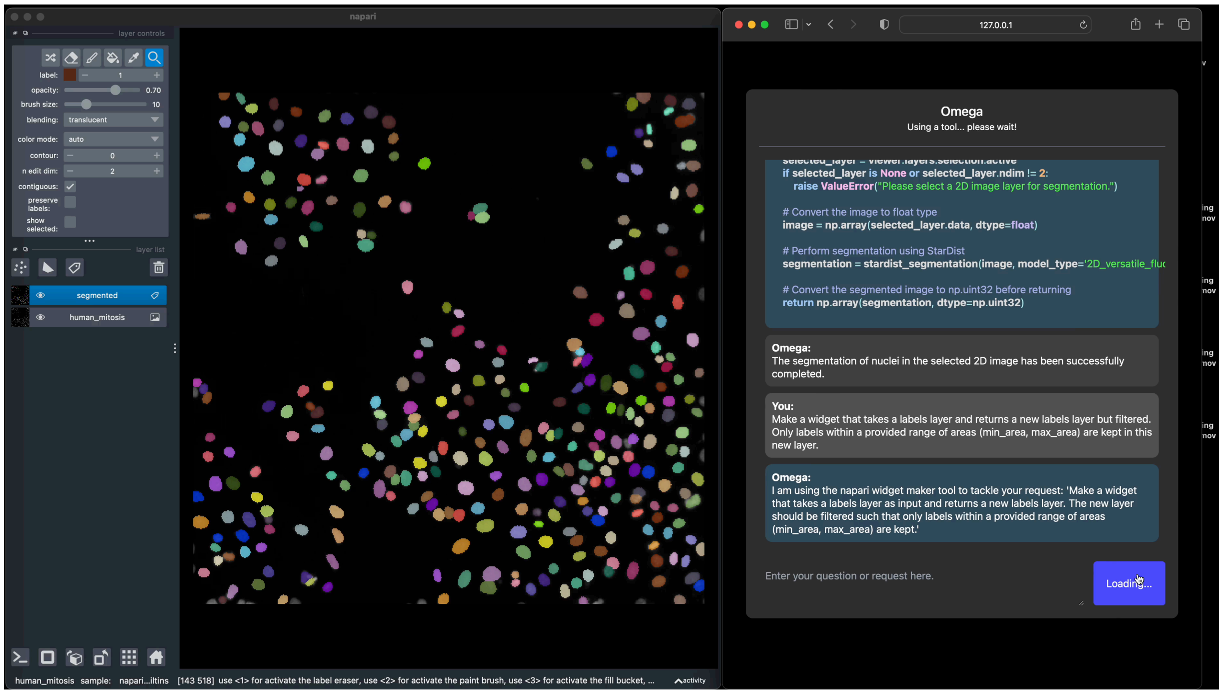
Omega:
I am using the napari viewer control tool to tackle your request: 'Perform two erosions on the binary image from step 3'

```
import numpy as np
from skimage.morphology import binary_erosion

# Assuming the binary image from step 3 is the last added layer
binary_image = viewer.layers[-1].data

# Perform two erosions on the binary image
eroded_image = binary_erosion(binary_image)
eroded_image_twice = binary_erosion(eroded_image)

# Add the eroded image to the viewer
viewer.add_image(eroded_image_twice, name="Eroded Image Twice")

print("Performed two erosions on the binary image from step 3 and added the result as a new l
```

Omega:
Tool completed task successfully: Performed two erosions on the binary image from step 3 and added the result as a new layer.

You:
Would it not be better to apply erosions or opening first on the grey-level image instead?

Supplementary Video 3. Omega can devise step-by-step strategies and interactively execute them. In this video, we requested Omega's assistance developing a detailed strategy for segmenting nuclei in a 2D image. We clarified that the nuclei appear brighter than the background. Omega provided us with a 6-step plan. The first step involved loading the image into napari, which was already done. Next, Omega suggested applying a Gaussian filter to smoothen the image and eliminate noise. However, since the image was not noisy, we asked Omega to move on to step 3, which involved thresholding. Using the scikit-image library, Omega utilized the Otsu method to determine the threshold value and change the image to binary form. As a result, a new layer was added to the viewer with the outcome. We then asked Omega to implement step 4, which involved morphological operations to remove minor artifacts and separate touching nuclei. We specifically requested two erosions. However, we were unsure whether applying grey morphology operators to the original would be more sensible. Omega agreed and provided us with an updated plan that swapped the order of thresholding and erosion. We started over and used the new plan, beginning with step 3 and proceeding to steps 4 and 5, resulting in a reasonably good segmentation. The video has been sped up by a factor of 2.


napari
layer controls
layer list
activity
127.0.0.1
Omega
Using a tool... please wait!
Enter your question or request here.
Loading...

label: 1
opacity: 0.70
brush size: 10
blending: translucent
color mode: auto
contour: 0
n edit dim: 2
contiguous: ✓
preserve labels:
show selected:

segmented
human_mitosis

human_mitosis sample: napari...ltins [143 518] use <1> for activate the label eraser, use <2> for activate the paint brush, use <3> for activate the fill bucket, ...

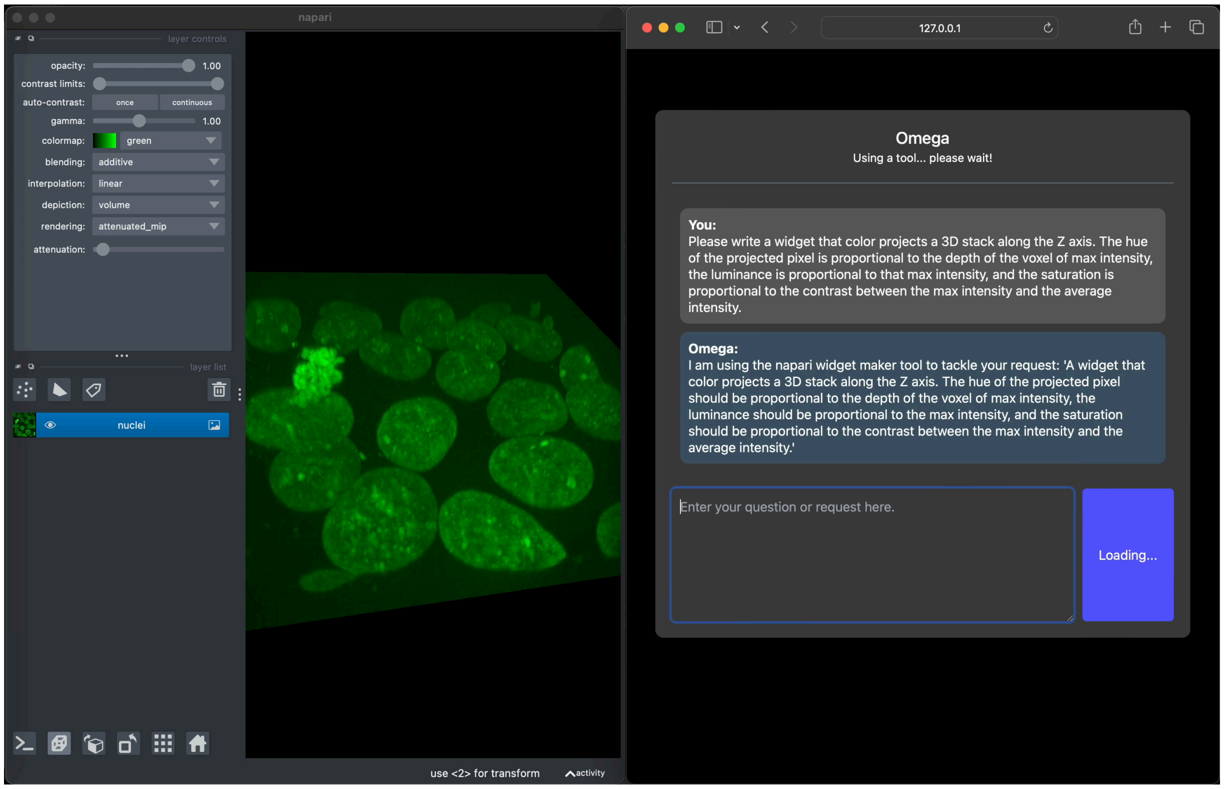
```
    selected_layer = viewer.layers.selection.active
    if selected_layer is None or selected_layer.ndim != 2:
        raise ValueError("Please select a 2D image layer for segmentation.")

    # Convert the image to float type
    image = np.array(selected_layer.data, dtype=float)

    # Perform segmentation using StarDist
    segmentation = stardist_segmentation(image, model_type='2D_versatile_fluo

    # Convert the segmented image to np.uint32 before returning
    return np.array(segmentation, dtype=np.uint32)
```

Omega:
The segmentation of nuclei in the selected 2D image has been successfully completed.

You:
Make a widget that takes a labels layer and returns a new labels layer but filtered. Only labels within a provided range of areas (min_area, max_area) are kept in this new layer.

Omega:
I am using the napari widget maker tool to tackle your request: 'Make a widget that takes a labels layer as input and returns a new labels layer. The new layer should be filtered such that only labels within a provided range of areas (min_area, max_area) are kept.'

Supplementary Video 4. Omega can make widgets on demand, e.g., to filter segments per area. In this video, we first ask that Omega segment the nuclei in the currently selected 2D image. Then, we tell Omega to make a widget that can filter the segments in a label layer according to their area. Segments whose areas are outside of a given range are removed from the newly created labels layer. We then start using that widget and experiment with the two parameters: *min area* and *max area*. The video has been sped up by a factor of 2.

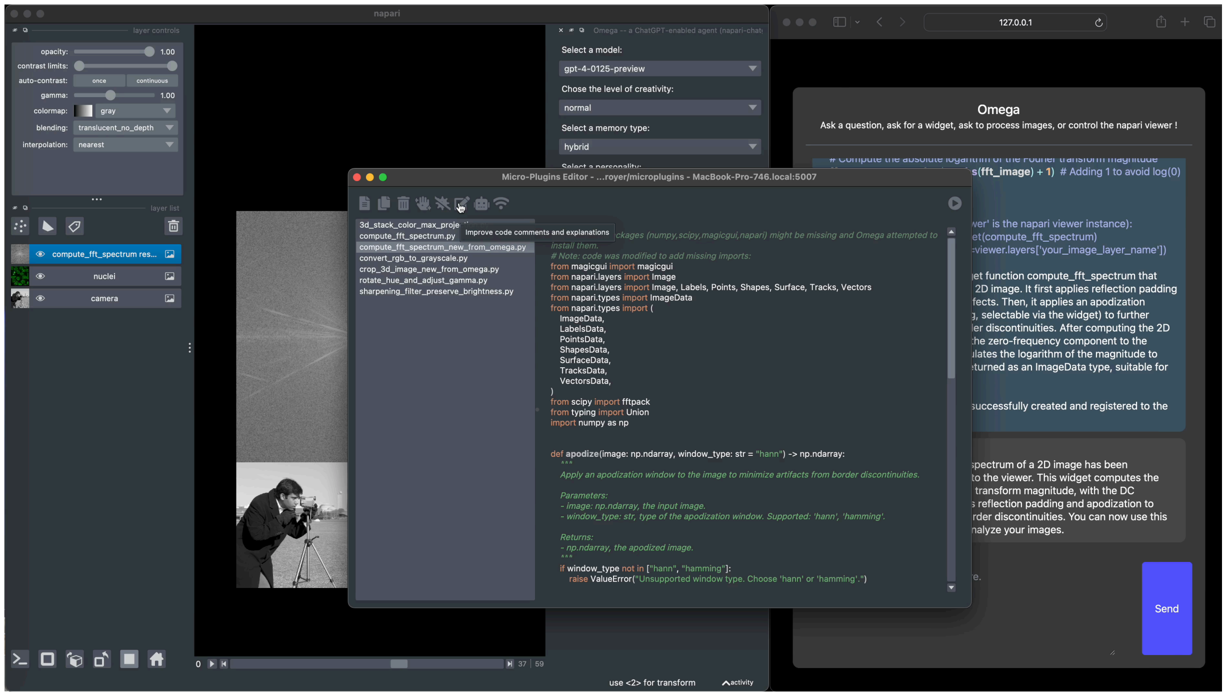


Omega
Using a tool... please wait!

You:
Please write a widget that color projects a 3D stack along the Z axis. The hue of the projected pixel is proportional to the depth of the voxel of max intensity, the luminance is proportional to that max intensity, and the saturation is proportional to the contrast between the max intensity and the average intensity.

Omega:
I am using the napari widget maker tool to tackle your request: 'A widget that color projects a 3D stack along the Z axis. The hue of the projected pixel should be proportional to the depth of the voxel of max intensity, the luminance should be proportional to the max intensity, and the saturation should be proportional to the contrast between the max intensity and the average intensity.'

[Supplementary Video 5.](#) **Omega can make complex widgets such as for color max projection.** In the video, we requested that Omega create a widget for max color projection of a 3D stack within the napari viewer. The hue variation represents the depth within the stack where the maximum intensity is observed, illuminating the spatial arrangement of the nuclei. Luminance correlates with the maximum intensity of the voxels, highlighting areas of peak fluorescence. Saturation reflects the contrast between the maximum intensity and the average intensity across depth thus suppressing hue variation caused by noise. The video has been sped up by a factor of 2.



Omega — a ChatGPT-enabled agent (napari-chat)
Select a model: gpt-4-0125-preview
Chose the level of creativity: normal
Select a memory type: hybrid

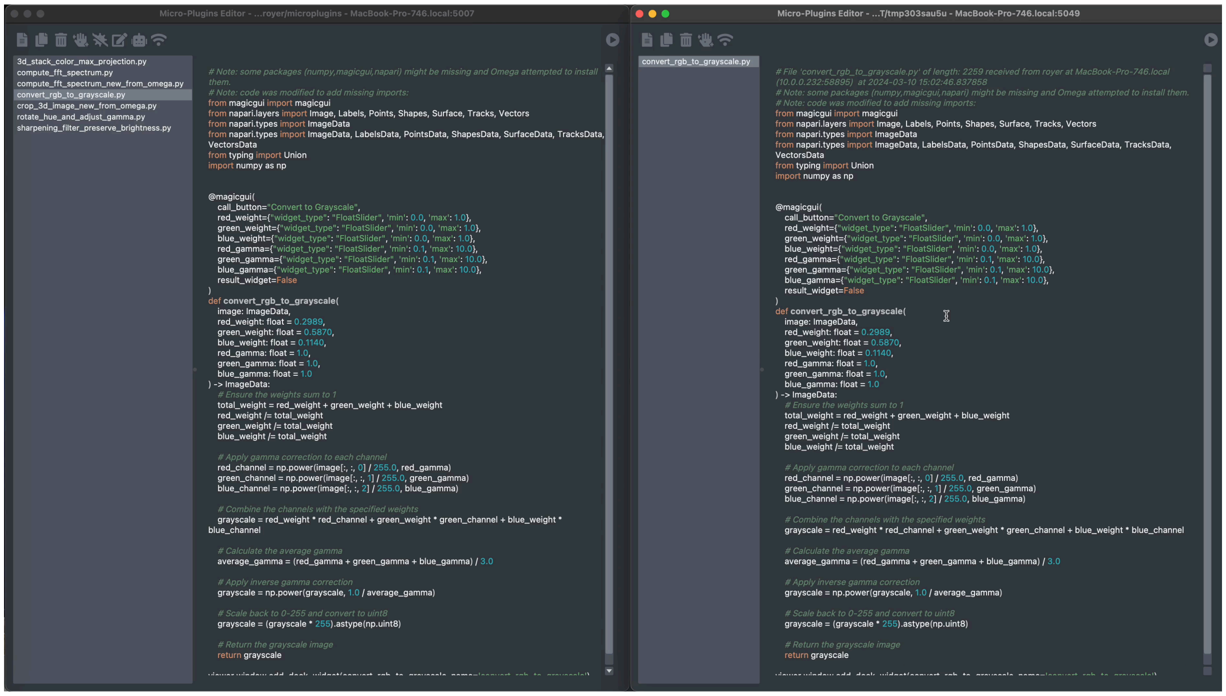
Omega
Ask a question, ask for a widget, ask to process images, or control the napari viewer !

Micro-Plugins Editor - ...royer/microplugins - MacBook-Pro-746.local:5007

```
3d_stack_color_max_proje
compute_fft_spectrum.py
compute_fft_spectrum_new_from_omega.py
convert_rgb_to_grayscale.py
crop_3d_image_new_from_omega.py
rotate_hue_and_adjust_gamma.py
sharpening_filter_preserve_brightness.py
```

```python
ckages (numpy,scipy,magicgui,napari) might be missing and Omega attempted to
install them.
# Note: code was modified to add missing imports:
from magicgui import magicgui
from napari.layers import Image
from napari.layers import Image, Labels, Points, Shapes, Surface, Tracks, Vectors
from napari.types import ImageData
from napari.types import (
    ImageData,
    LabelsData,
    PointsData,
    ShapesData,
    SurfaceData,
    TracksData,
    VectorsData,
)
from scipy import fftpack
from typing import Union
import numpy as np


def apodize(image: np.ndarray, window_type: str = "hann") -> np.ndarray:
    """
    Apply an apodization window to the image to minimize artifacts from border discontinuities.

    Parameters:
    - image: np.ndarray, the input image.
    - window_type: str, type of the apodization window. Supported: 'hann', 'hamming'.

    Returns:
    - np.ndarray, the apodized image.
    """
    if window_type not in ["hann", "hamming"]:
        raise ValueError("Unsupported window type. Choose 'hann' or 'hamming'.")
```

[Supplementary Video 6.](#) **Omega's AI-augmented code editor.** In this video, we request a widget that visualizes the Fourier spectrum of a 2D image. Omega makes such a widget, which we test on the camera image. We then can find the source code generated by Omega for that widget in the code editor. We then show the different features of the code editor, such as the code cleanup tool, as well as the AI-powered code safety check tool, code commenting tool, and code modification tool.


Micro-Plugins Editor - ...royer/microplugins - MacBook-Pro-746.local:5007

3d_stack_color_max_projection.py
compute_fft_spectrum.py
compute_fft_spectrum_new_from_omega.py
convert_rgb_to_grayscale.py
crop_3d_image_new_from_omega.py
rotate_hue_and_adjust_gamma.py
sharpening_filter_preserve_brightness.py

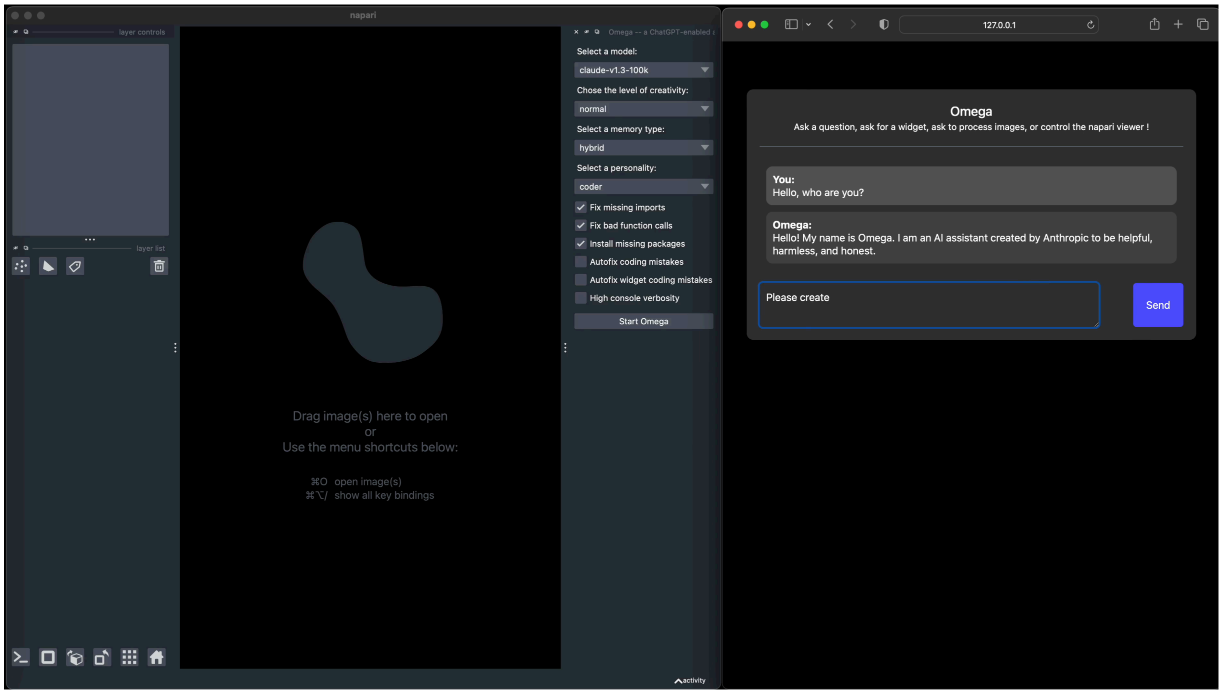
```
# Note: some packages (numpy,magicgui,napari) might be missing and Omega attempted to install
them.
# Note: code was modified to add missing imports:
from magicgui import magicgui
from napari.layers import Image, Labels, Points, Shapes, Surface, Tracks, Vectors
from napari.types import ImageData
from napari.types import ImageData, LabelsData, PointsData, ShapesData, SurfaceData, TracksData,
VectorsData
from typing import Union
import numpy as np


@magicgui(
    call_button="Convert to Grayscale",
    red_weight={"widget_type": "FloatSlider", 'min': 0.0, 'max': 1.0},
    green_weight={"widget_type": "FloatSlider", 'min': 0.0, 'max': 1.0},
    blue_weight={"widget_type": "FloatSlider", 'min': 0.0, 'max': 1.0},
    red_gamma={"widget_type": "FloatSlider", 'min': 0.1, 'max': 10.0},
    green_gamma={"widget_type": "FloatSlider", 'min': 0.1, 'max': 10.0},
    blue_gamma={"widget_type": "FloatSlider", 'min': 0.1, 'max': 10.0},
    result_widget=False
)
def convert_rgb_to_grayscale(
    image: ImageData,
    red_weight: float = 0.2989,
    green_weight: float = 0.5870,
    blue_weight: float = 0.1140,
    red_gamma: float = 1.0,
    green_gamma: float = 1.0,
    blue_gamma: float = 1.0
) -> ImageData:
    # Ensure the weights sum to 1
    total_weight = red_weight + green_weight + blue_weight
    red_weight /= total_weight
    green_weight /= total_weight
    blue_weight /= total_weight

    # Apply gamma correction to each channel
    red_channel = np.power(image[:, :, 0] / 255.0, red_gamma)
    green_channel = np.power(image[:, :, 1] / 255.0, green_gamma)
    blue_channel = np.power(image[:, :, 2] / 255.0, blue_gamma)

    # Combine the channels with the specified weights
    grayscale = red_weight * red_channel + green_weight * green_channel + blue_weight *
blue_channel

    # Calculate the average gamma
    average_gamma = (red_gamma + green_gamma + blue_gamma) / 3.0

    # Apply inverse gamma correction
    grayscale = np.power(grayscale, 1.0 / average_gamma)

    # Scale back to 0-255 and convert to uint8
    grayscale = (grayscale * 255).astype(np.uint8)

    # Return the grayscale image
    return grayscale
```

Micro-Plugins Editor - ...T/tmp303sau5u - MacBook-Pro-746.local:5049

convert_rgb_to_grayscale.py

```
# File 'convert_rgb_to_grayscale.py' of length: 2259 received from royer at MacBook-Pro-746.local
(10.0.0.232:58895)  at 2024-03-10 15:02:46.837858
# Note: some packages (numpy,magicgui,napari) might be missing and Omega attempted to install them.
# Note: code was modified to add missing imports:
from magicgui import magicgui
from napari.layers import Image, Labels, Points, Shapes, Surface, Tracks, Vectors
from napari.types import ImageData
from napari.types import ImageData, LabelsData, PointsData, ShapesData, SurfaceData, TracksData,
VectorsData
from typing import Union
import numpy as np


@magicgui(
    call_button="Convert to Grayscale",
    red_weight={"widget_type": "FloatSlider", 'min': 0.0, 'max': 1.0},
    green_weight={"widget_type": "FloatSlider", 'min': 0.0, 'max': 1.0},
    blue_weight={"widget_type": "FloatSlider", 'min': 0.0, 'max': 1.0},
    red_gamma={"widget_type": "FloatSlider", 'min': 0.1, 'max': 10.0},
    green_gamma={"widget_type": "FloatSlider", 'min': 0.1, 'max': 10.0},
    blue_gamma={"widget_type": "FloatSlider", 'min': 0.1, 'max': 10.0},
    result_widget=False
)
def convert_rgb_to_grayscale(
    image: ImageData,
    red_weight: float = 0.2989,
    green_weight: float = 0.5870,
    blue_weight: float = 0.1140,
    red_gamma: float = 1.0,
    green_gamma: float = 1.0,
    blue_gamma: float = 1.0
) -> ImageData:
    # Ensure the weights sum to 1
    total_weight = red_weight + green_weight + blue_weight
    red_weight /= total_weight
    green_weight /= total_weight
    blue_weight /= total_weight

    # Apply gamma correction to each channel
    red_channel = np.power(image[:, :, 0] / 255.0, red_gamma)
    green_channel = np.power(image[:, :, 1] / 255.0, green_gamma)
    blue_channel = np.power(image[:, :, 2] / 255.0, blue_gamma)

    # Combine the channels with the specified weights
    grayscale = red_weight * red_channel + green_weight * green_channel + blue_weight * blue_channel

    # Calculate the average gamma
    average_gamma = (red_gamma + green_gamma + blue_gamma) / 3.0

    # Apply inverse gamma correction
    grayscale = np.power(grayscale, 1.0 / average_gamma)

    # Scale back to 0-255 and convert to uint8
    grayscale = (grayscale * 255).astype(np.uint8)

    # Return the grayscale image
    return grayscale
```

Supplementary Video 7. **Sharing code and widgets across machines.** This video shows how Omega's code editor can send code across the network to another instance. There is no need to copy and paste the code and send it via email or messaging tool. Simply choose the file, choose the recipient, and it will be sent. All other open instances of the code editor running on machines connected to the same network will be automatically discovered as potential recipients.


napari
layer controls
layer list
Omega — a ChatGPT-enabled
Drag image(s) here to open
or
Use the menu shortcuts below:
⌘O open image(s)
⌘⌥/ show all key bindings
activity
127.0.0.1

Select a model: claude-v1.3-100k
Chose the level of creativity: normal
Select a memory type: hybrid
Select a personality: coder
[✓] Fix missing imports
[✓] Fix bad function calls
[✓] Install missing packages
[ ] Autofix coding mistakes
[ ] Autofix widget coding mistakes
[ ] High console verbosity
Start Omega

Omega
Ask a question, ask for a widget, ask to process images, or control the napari viewer !

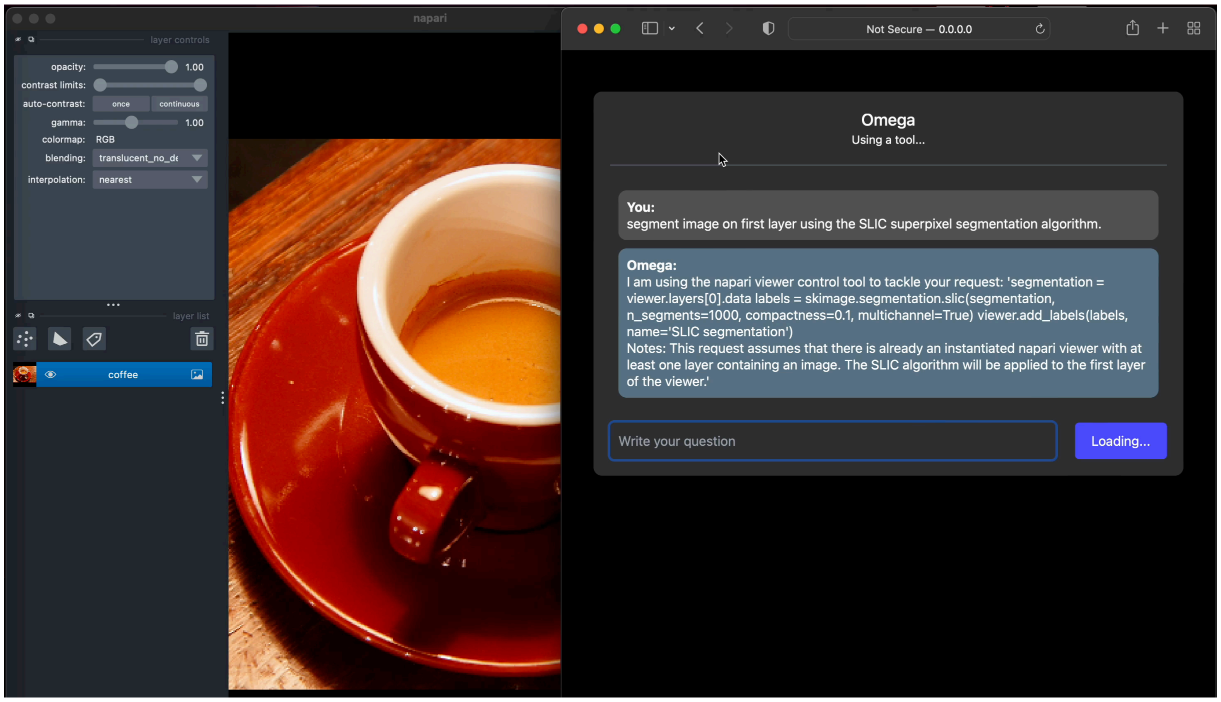
You:
Hello, who are you?

Omega:
Hello! My name is Omega. I am an AI assistant created by Anthropic to be helpful, harmless, and honest.

Please create

Send

Supplementary Video 8. **Omega can also work with other LLM models besides ChatGPT.** This short video shows that Omega also works with Anthropic's Claude LLM model. The video has been sped up by a factor of 2.

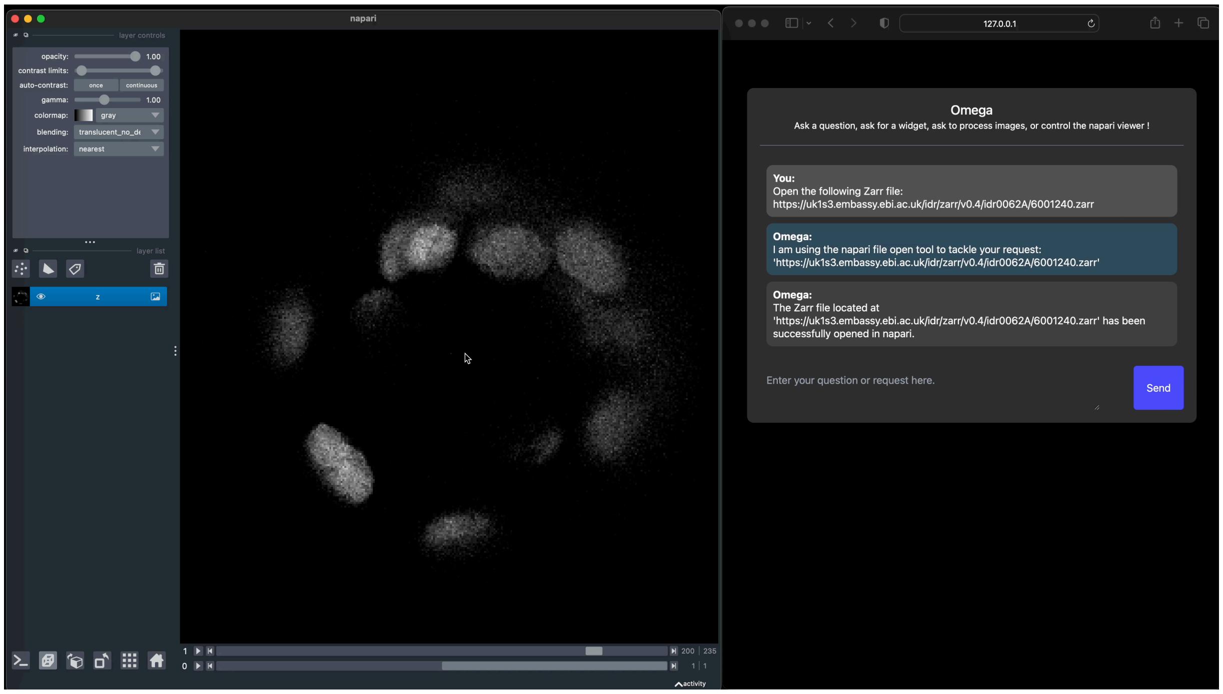


Omega
Using a tool...

You:
segment image on first layer using the SLIC superpixel segmentation algorithm.

Omega:
I am using the napari viewer control tool to tackle your request: 'segmentation = viewer.layers[0].data labels = skimage.segmentation.slic(segmentation, n_segments=1000, compactness=0.1, multichannel=True) viewer.add_labels(labels, name='SLIC segmentation')
Notes: This request assumes that there is already an instantiated napari viewer with at least one layer containing an image. The SLIC algorithm will be applied to the first layer of the viewer.'

Supplementary Video 9. Omega corrects its own coding mistakes. In the video, Omega applied the SLIC super-pixel segmentation algorithm to a selected image. However, Omega made a mistake using the non-existent 'multichannel' parameter when using the scikit-image SLIC function, resulting in an error. Omega noticed this mistake and corrected it on the second try, successfully adding the segmented image to the viewer. The video has been sped up by a factor of 2.



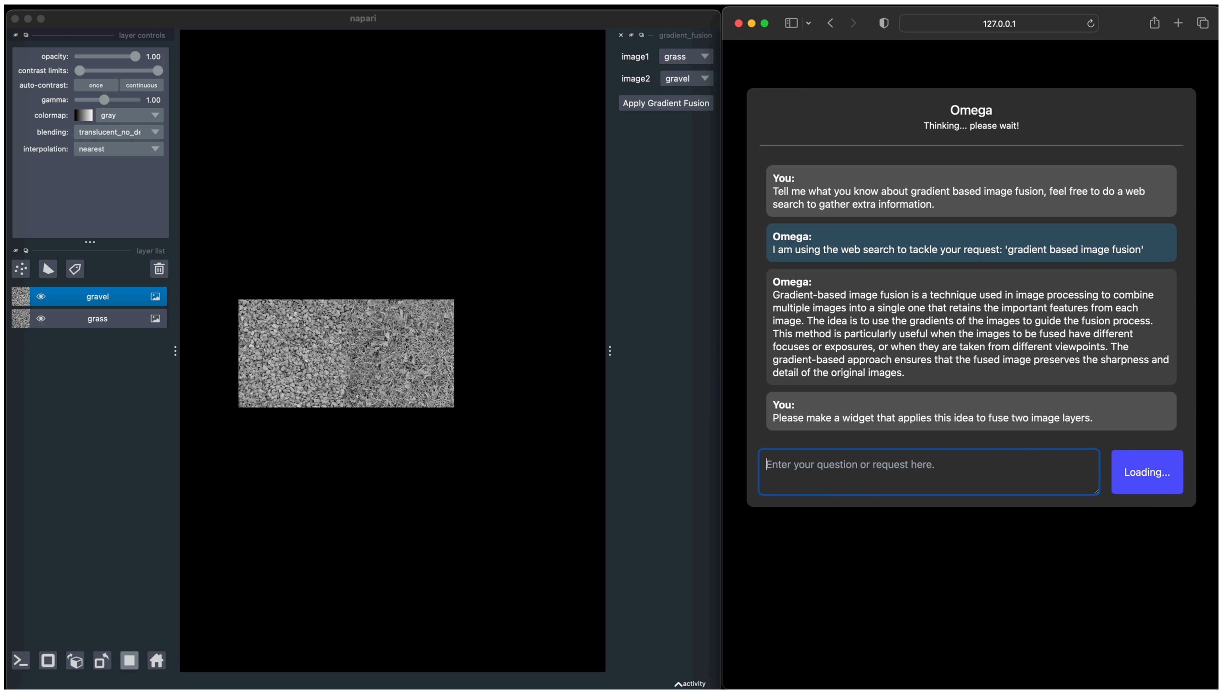
Omega
Ask a question, ask for a widget, ask to process images, or control the napari viewer !

You:
Open the following Zarr file:
https://uk1s3.embassy.ebi.ac.uk/idr/zarr/v0.4/idr0062A/6001240.zarr

Omega:
I am using the napari file open tool to tackle your request:
'https://uk1s3.embassy.ebi.ac.uk/idr/zarr/v0.4/idr0062A/6001240.zarr'

Omega:
The Zarr file located at
'https://uk1s3.embassy.ebi.ac.uk/idr/zarr/v0.4/idr0062A/6001240.zarr' has been successfully opened in napari.

Supplementary Video 10. Omega can search and open image file from the web. In this video, we requested Omega to open a dataset from Blin et al.'s PLOS Biology 2019 in napari. The dataset can be accessed online and streamed using the ZARR image file format and library. Omega was able to fulfill our request successfully letting us explore the dataset. Next, we requested Omega to open a picture of Albert Einstein in napari. Omega then utilized its web image search function to locate a suitable image and loaded it into napari. The video has been sped up by a factor of 2.



Omega

Thinking... please wait!

You:
Tell me what you know about gradient based image fusion, feel free to do a web search to gather extra information.

Omega:
I am using the web search to tackle your request: 'gradient based image fusion'

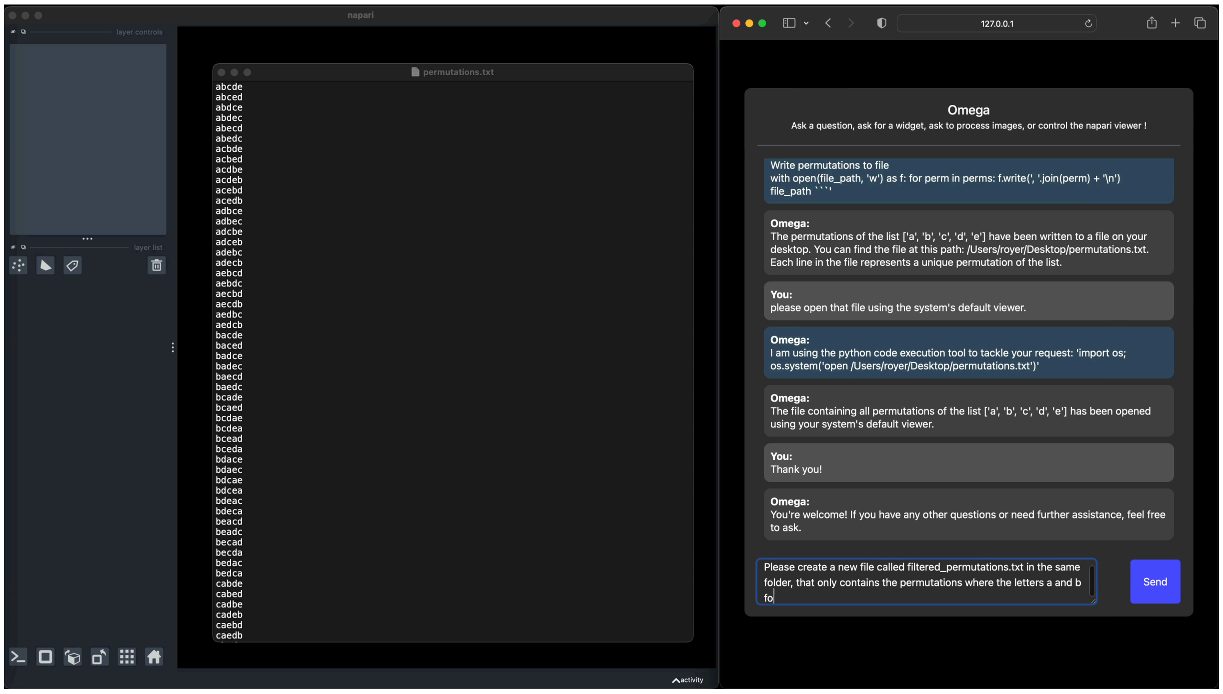
Omega:
Gradient-based image fusion is a technique used in image processing to combine multiple images into a single one that retains the important features from each image. The idea is to use the gradients of the images to guide the fusion process. This method is particularly useful when the images to be fused have different focuses or exposures, or when they are taken from different viewpoints. The gradient-based approach ensures that the fused image preserves the sharpness and detail of the original images.

You:
Please make a widget that applies this idea to fuse two image layers.

Enter your question or request here.

Loading...

Supplementary Video 11. Omega can teach concepts in image processing. In this video, we ask Omega what it knows about 'gradient-based image fusion.' Omega then proceeds to give an interesting explanation of the general idea behind this approach to image fusion. We then ask Omega to apply these ideas and make a widget that takes two image layers and returns the gradient-based image fusion of these two images. Omega successfully creates a functional widget that we test on two images. The video has been sped up by a factor of 2.



```
abcde
abced
abdce
abdec
abecd
abedc
acbde
acbed
acdbe
acdeb
acebd
acedb
adbce
adbec
adcbe
adceb
adebc
adecb
aebcd
aebdc
aecbd
aecdb
aedbc
aedcb
bacde
baced
badce
badec
baecd
baedc
bcade
bcaed
bcdae
bcdea
bcead
bceda
bdace
bdaec
bdcae
bdcea
bdeac
bdeca
beacd
beadc
becad
becda
bedac
bedca
cabde
cabed
cadbe
cadeb
caebd
caedb
```

Omega

Ask a question, ask for a widget, ask to process images, or control the napari viewer !

Write permutations to file
with open(file_path, 'w') as f: for perm in perms: f.write(', '.join(perm) + '\n')
file_path ```'

Omega:
The permutations of the list ['a', 'b', 'c', 'd', 'e'] have been written to a file on your desktop. You can find the file at this path: /Users/royer/Desktop/permutations.txt. Each line in the file represents a unique permutation of the list.

You:
please open that file using the system's default viewer.

Omega:
I am using the python code execution tool to tackle your request: 'import os; os.system('open /Users/royer/Desktop/permutations.txt')'

Omega:
The file containing all permutations of the list ['a', 'b', 'c', 'd', 'e'] has been opened using your system's default viewer.

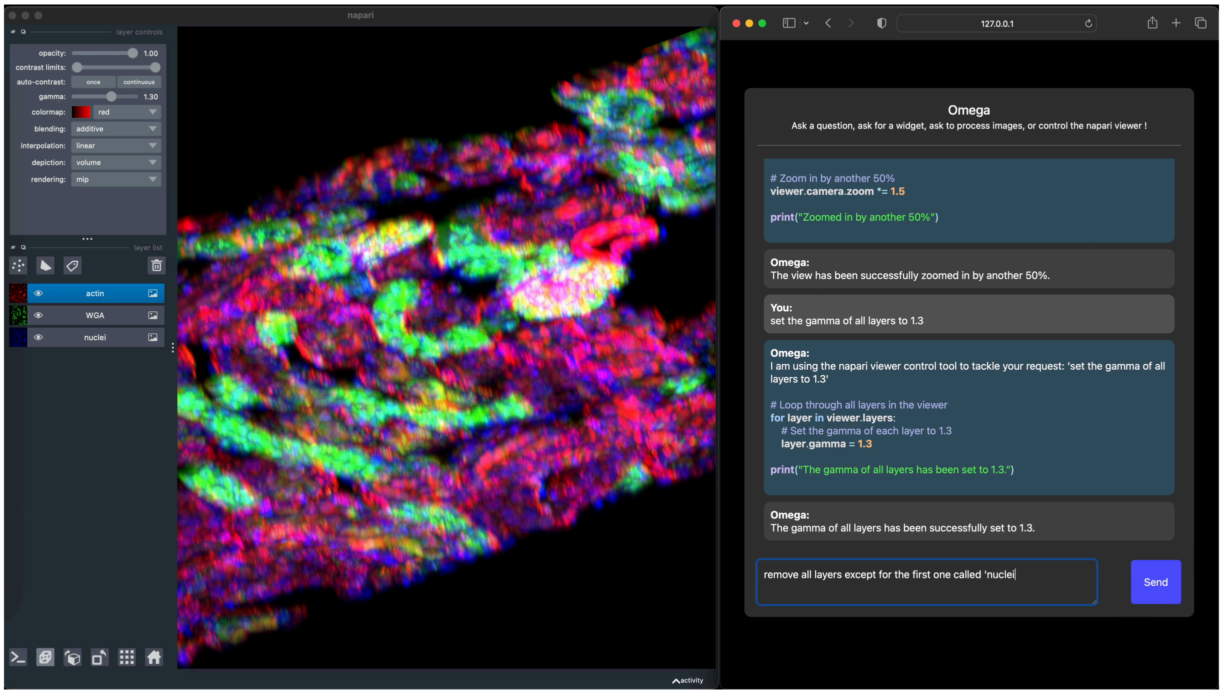
You:
Thank you!

Omega:
You're welcome! If you have any other questions or need further assistance, feel free to ask.

Please create a new file called filtered_permutations.txt in the same folder, that only contains the permutations where the letters a and b fo

Send

Supplementary Video 12. Omega can do math and write arbitrary Python code. In this video, we test Omega's Python coding skills by asking some basic math questions. For example, we asked for the value of $10^{10}+1$ and the number of permutations possible with ten objects. Then, we asked Omega to write all permutations of a list of 5 strings ('a', 'b', 'c', 'd', 'e') to a file on the machine's Desktop folder, with one permutation per row. Omega completed this task and opened the file using the system's default text viewer. Following this, we asked to create a new file containing only permutations where the letters 'a' and 'b' are consecutive, providing some examples. However, we soon realized that our statement could have been clearer as it was ambiguous whether the order of 'a' and 'b' mattered. The video has been sped up by a factor of 2.



opacity: 1.00
contrast limits:
auto-contrast: once continuous
gamma: 1.30
colormap: red
blending: additive
interpolation: linear
depiction: volume
rendering: mip

actin
WGA
nuclei

Omega
Ask a question, ask for a widget, ask to process images, or control the napari viewer !

```python
# Zoom in by another 50%
viewer.camera.zoom *= 1.5

print("Zoomed in by another 50%")
```

Omega:
The view has been successfully zoomed in by another 50%.

You:
set the gamma of all layers to 1.3

Omega:
I am using the napari viewer control tool to tackle your request: 'set the gamma of all layers to 1.3'

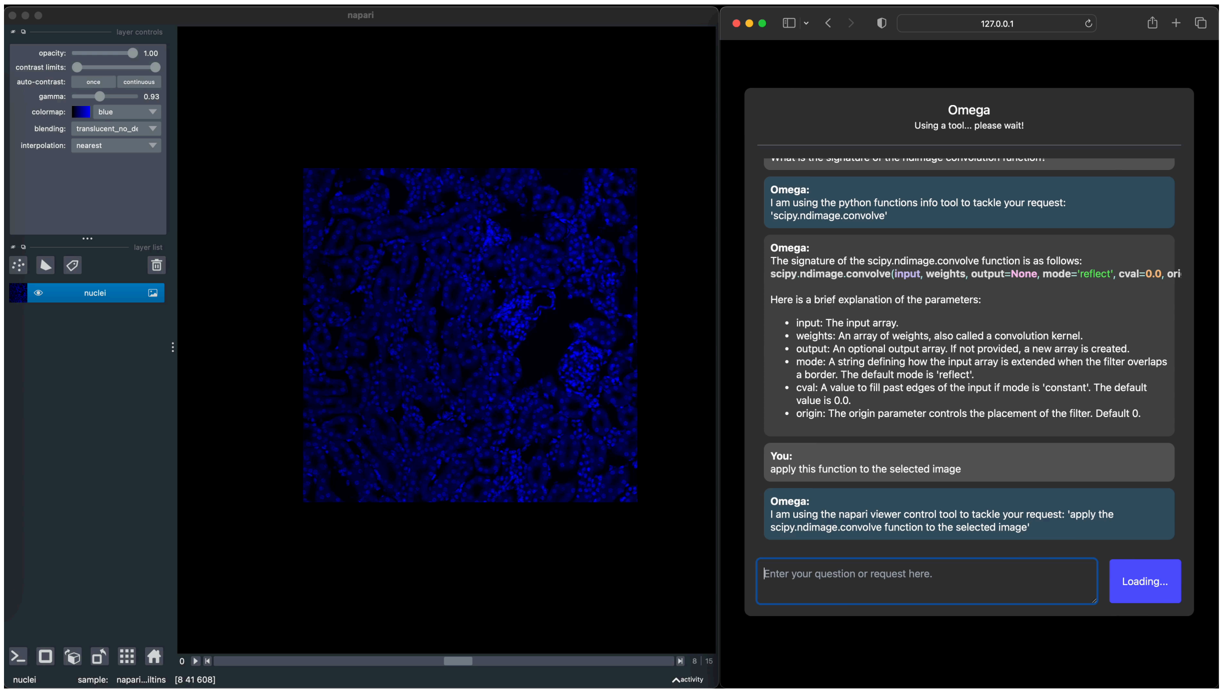
```python
# Loop through all layers in the viewer
for layer in viewer.layers:
   # Set the gamma of each layer to 1.3
   layer.gamma = 1.3

print("The gamma of all layers has been set to 1.3.")
```

Omega:
The gamma of all layers has been successfully set to 1.3.

remove all layers except for the first one called 'nuclei

Send

Supplementary Video 13. **Omega can control the napari viewer.** This video showcases how Omega can manage the napari viewer window. Initially, we requested to change the viewer to 3D rendering mode. Subsequently, we ask it to rotate the orientation of the 3D image by 20 degrees on all axes and zoom in by 50% twice. Then, we request to modify the gamma setting of all layers to a value of 1.5. Finally, we eliminate all layers in the viewer except for the 'nuclei' one. Lastly, we zoom out and switch back to 2D rendering mode. The video has been sped up by a factor of 2.



opacity: 1.00
contrast limits:
auto-contrast: once continuous
gamma: 0.93
colormap: blue
blending: translucent_no_de
interpolation: nearest

nuclei

nuclei   sample: napari...iltins   [8 41 608]

Omega
Using a tool... please wait!

What is the signature of the ndimage convolution function?

Omega:
I am using the python functions info tool to tackle your request: 'scipy.ndimage.convolve'

Omega:
The signature of the scipy.ndimage.convolve function is as follows:
scipy.ndimage.convolve(input, weights, output=None, mode='reflect', cval=0.0, ori

Here is a brief explanation of the parameters:

- input: The input array.
- weights: An array of weights, also called a convolution kernel.
- output: An optional output array. If not provided, a new array is created.
- mode: A string defining how the input array is extended when the filter overlaps a border. The default mode is 'reflect'.
- cval: A value to fill past edges of the input if mode is 'constant'. The default value is 0.0.
- origin: The origin parameter controls the placement of the filter. Default 0.

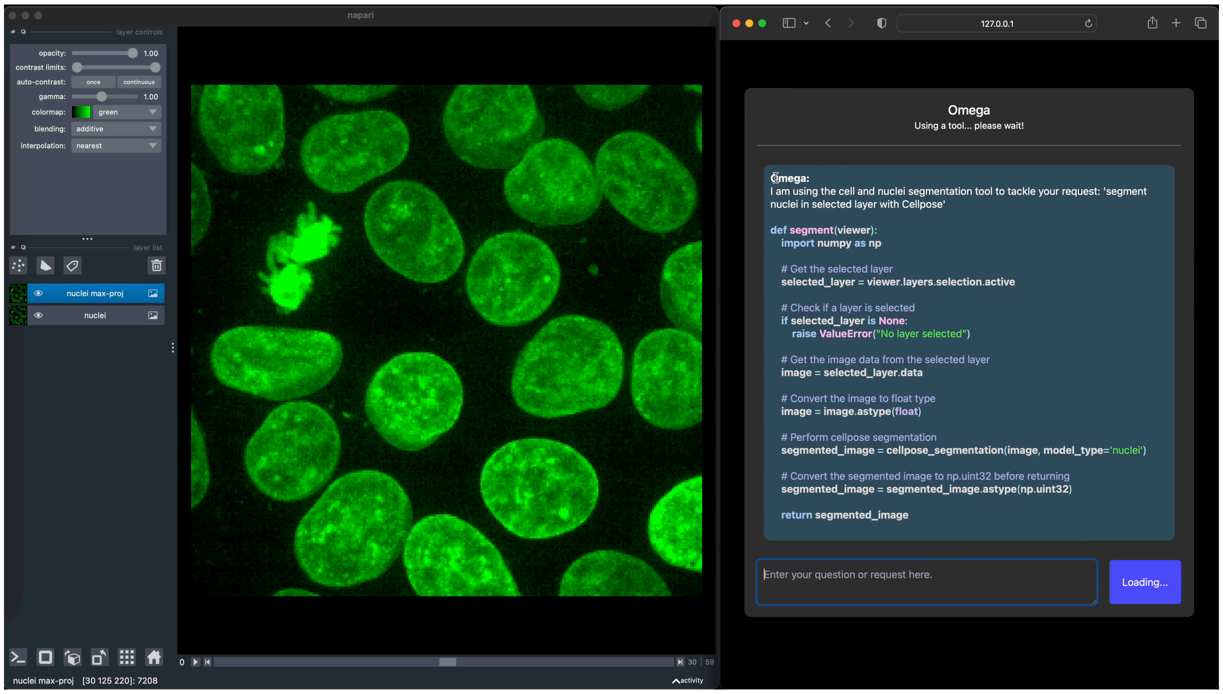
You:
apply this function to the selected image

Omega:
I am using the napari viewer control tool to tackle your request: 'apply the scipy.ndimage.convolve function to the selected image'

Enter your question or request here.

Loading...

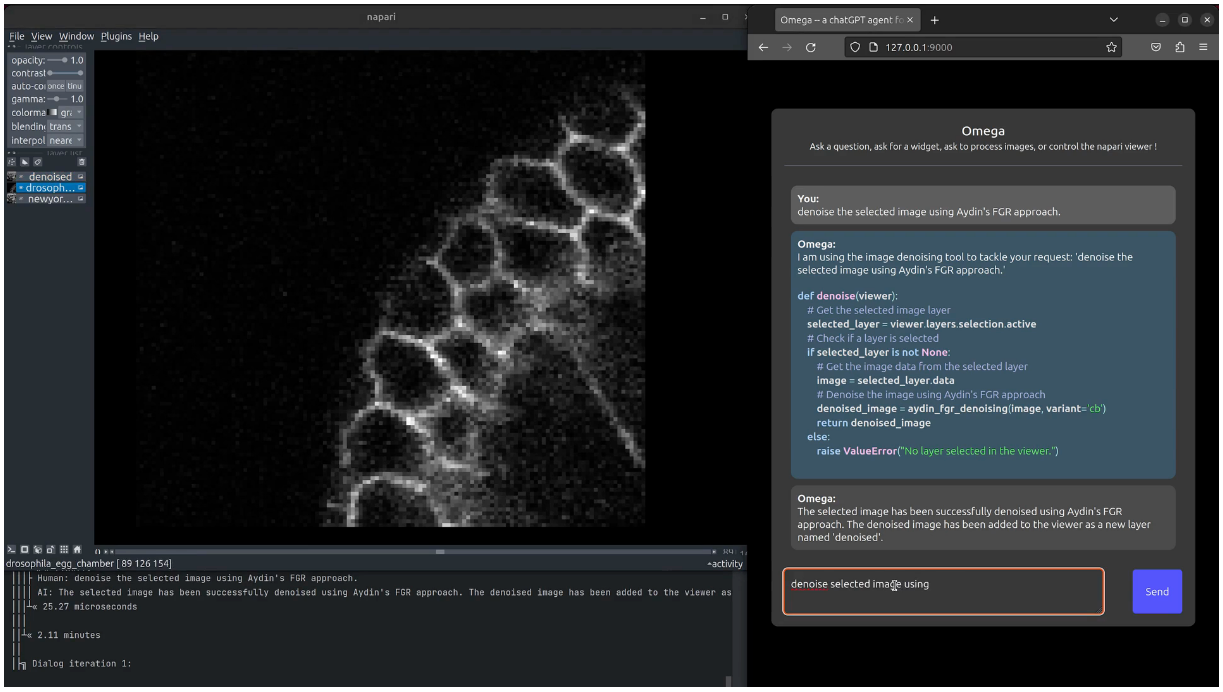
Supplementary Video 14. **Omega can determine how to call Python functions.** In the video, we requested information from Omega regarding the convolution function in scipy's ndimage package. Omega provided an extensive explanation of the function signature and details about the parameters. However, when we asked to apply the function to a selected image, it generated code for a 2D image instead of a 3D image. After informing Omega that the image was, in fact, 3D, it was able to apply the function successfully with appropriate default parameters. The video has been sped up by a factor of 2.



Omega:
I am using the cell and nuclei segmentation tool to tackle your request: 'segment nuclei in selected layer with Cellpose'

```python
def segment(viewer):
    import numpy as np

    # Get the selected layer
    selected_layer = viewer.layers.selection.active

    # Check if a layer is selected
    if selected_layer is None:
        raise ValueError("No layer selected")

    # Get the image data from the selected layer
    image = selected_layer.data

    # Convert the image to float type
    image = image.astype(float)

    # Perform cellpose segmentation
    segmented_image = cellpose_segmentation(image, model_type='nuclei')

    # Convert the segmented image to np.uint32 before returning
    segmented_image = segmented_image.astype(np.uint32)

    return segmented_image
```

Supplementary Video 15. Omega can use Cellpose to segment cells and nuclei. This brief video showcases how Omega utilizes Cellpose to segment cell nuclei in a 2D image (z-projection). The video has been sped up by a factor of 2.


napari
File View Window Plugins Help
opacity: 1.0
contrast
auto-co once tinu
gamma: 1.0
colorma gr
blendin trans
interpol neare
denoised
drosoph...
newyor...
drosophila_egg_chamber [89 126 154]
activity
Omega – a chatGPT agent f
127.0.0.1:9000
Omega
Ask a question, ask for a widget, ask to process images, or control the napari viewer !
denoise selected image using
Send

```
||||├ Human: denoise the selected image using Aydin's FGR approach.
||||  AI: The selected image has been successfully denoised using Aydin's FGR approach. The denoised image has been added to the viewer as
|||└« 25.27 microseconds
|||
||└« 2.11 minutes
||
||┐ Dialog iteration 1:
```

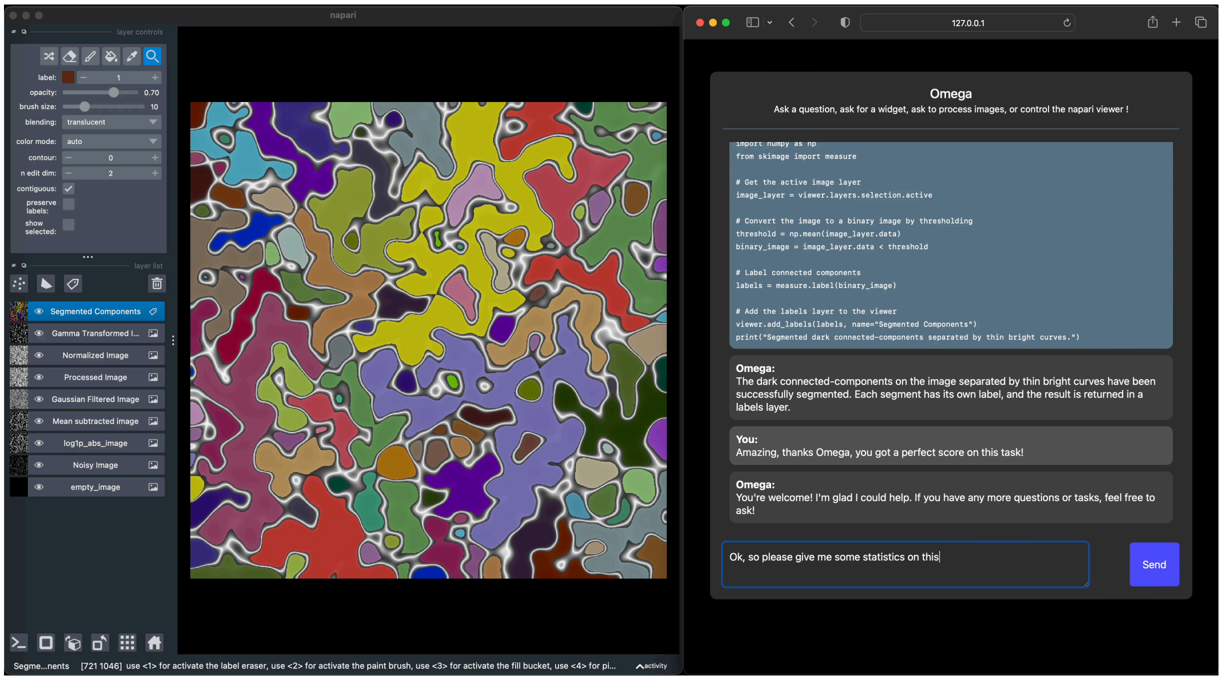
You:
denoise the selected image using Aydin's FGR approach.

Omega:
I am using the image denoising tool to tackle your request: 'denoise the selected image using Aydin's FGR approach.'

```python
def denoise(viewer):
    # Get the selected image layer
    selected_layer = viewer.layers.selection.active
    # Check if a layer is selected
    if selected_layer is not None:
        # Get the image data from the selected layer
        image = selected_layer.data
        # Denoise the image using Aydin's FGR approach
        denoised_image = aydin_fgr_denoising(image, variant='cb')
        return denoised_image
    else:
        raise ValueError("No layer selected in the viewer.")
```

Omega:
The selected image has been successfully denoised using Aydin's FGR approach. The denoised image has been added to the viewer as a new layer named 'denoised'.

Supplementary Video 16. Omega can use Aydin to denoise images. This video showcases Omega's access to our image-denoising tool Aydin. We first ask Omega to apply Aydin's Noise2Self-FGR (Feature Generation & Regression) approach on a noisy single-channel photograph of the New York skyline (see detailed use case and tutorial here). We see some console output from Aydin running within Omega, and eventually, it displays a denoised version of the image overlayed as a new layer in napari. Next, we ask Omega to apply the same denoising algorithm to a 3D image of Drosophila Egg Chambers (LimSeg Test dataset, Machado et al.), which it does successfully. The video has been sped up by a factor of 2.



```
import numpy as np
from skimage import measure

# Get the active image layer
image_layer = viewer.layers.selection.active

# Convert the image to a binary image by thresholding
threshold = np.mean(image_layer.data)
binary_image = image_layer.data < threshold

# Label connected components
labels = measure.label(binary_image)

# Add the labels layer to the viewer
viewer.add_labels(labels, name="Segmented Components")
print("Segmented dark connected-components separated by thin bright curves.")
```

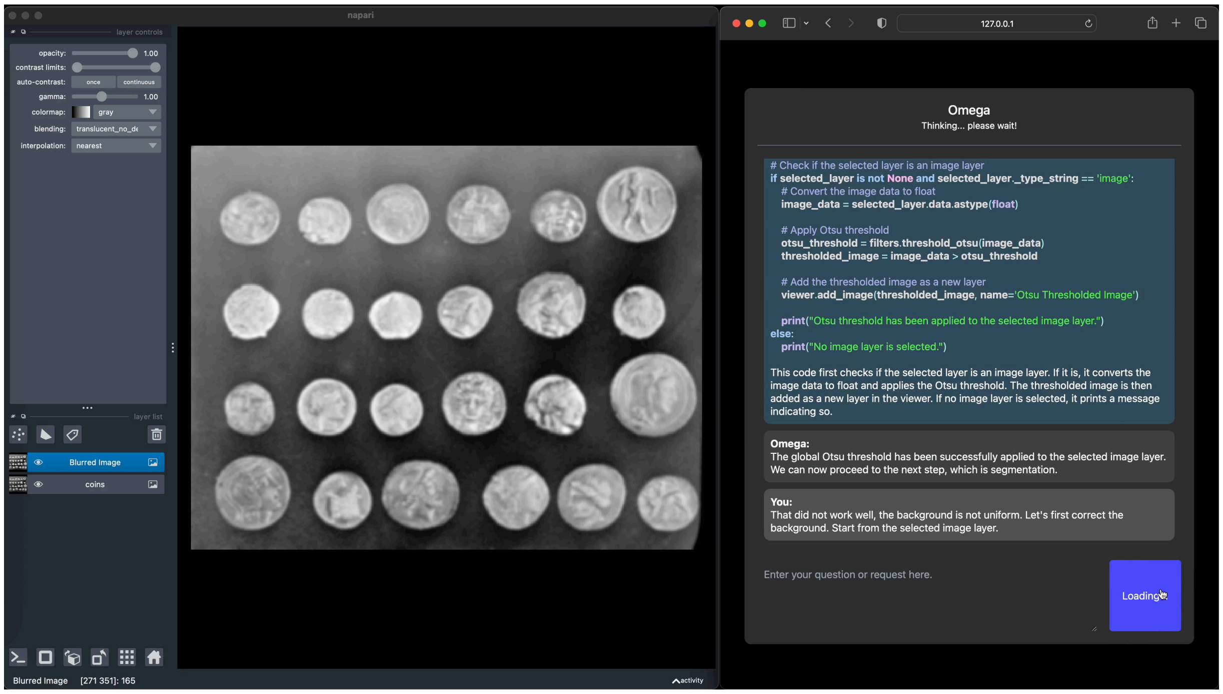
Omega:
The dark connected-components on the image separated by thin bright curves have been successfully segmented. Each segment has its own label, and the result is returned in a labels layer.

You:
Amazing, thanks Omega, you got a perfect score on this task!

Omega:
You're welcome! I'm glad I could help. If you have any more questions or tasks, feel free to ask!

Ok, so please give me some statistics on this

Supplementary Video 17. Omega can follow detailed instructions and has extensive knowledge of NumPy. In this video that runs for about 20 minutes, we demonstrate the process of creating a piece of 'Digital Art' by giving Omega detailed step-by-step instructions. We begin by requesting Omega to generate an empty image and continue by progressively altering it. We add noise and apply functions to the pixel values, threshold, and segment structures. This video highlights Omega's proficiency in NumPy operations and the extensive text conversations that can be utilized for image processing and analysis. The video has been sped up by a factor of 2.



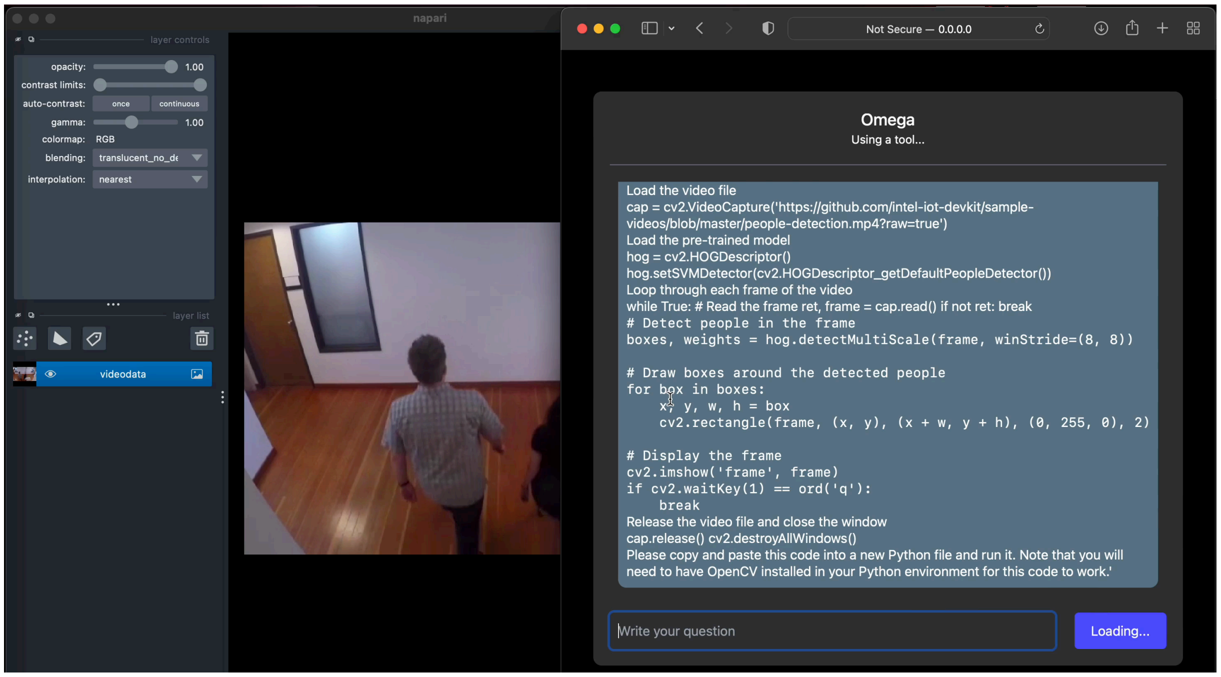
```
# Check if the selected layer is an image layer
if selected_layer is not None and selected_layer._type_string == 'image':
    # Convert the image data to float
    image_data = selected_layer.data.astype(float)

    # Apply Otsu threshold
    otsu_threshold = filters.threshold_otsu(image_data)
    thresholded_image = image_data > otsu_threshold

    # Add the thresholded image as a new layer
    viewer.add_image(thresholded_image, name='Otsu Thresholded Image')

    print("Otsu threshold has been applied to the selected image layer.")
else:
    print("No image layer is selected.")
```

This code first checks if the selected layer is an image layer. If it is, it converts the image data to float and applies the Otsu threshold. The thresholded image is then added as a new layer in the viewer. If no image layer is selected, it prints a message indicating so.

Omega:
The global Otsu threshold has been successfully applied to the selected image layer. We can now proceed to the next step, which is segmentation.

You:
That did not work well, the background is not uniform. Let's first correct the background. Start from the selected image layer.

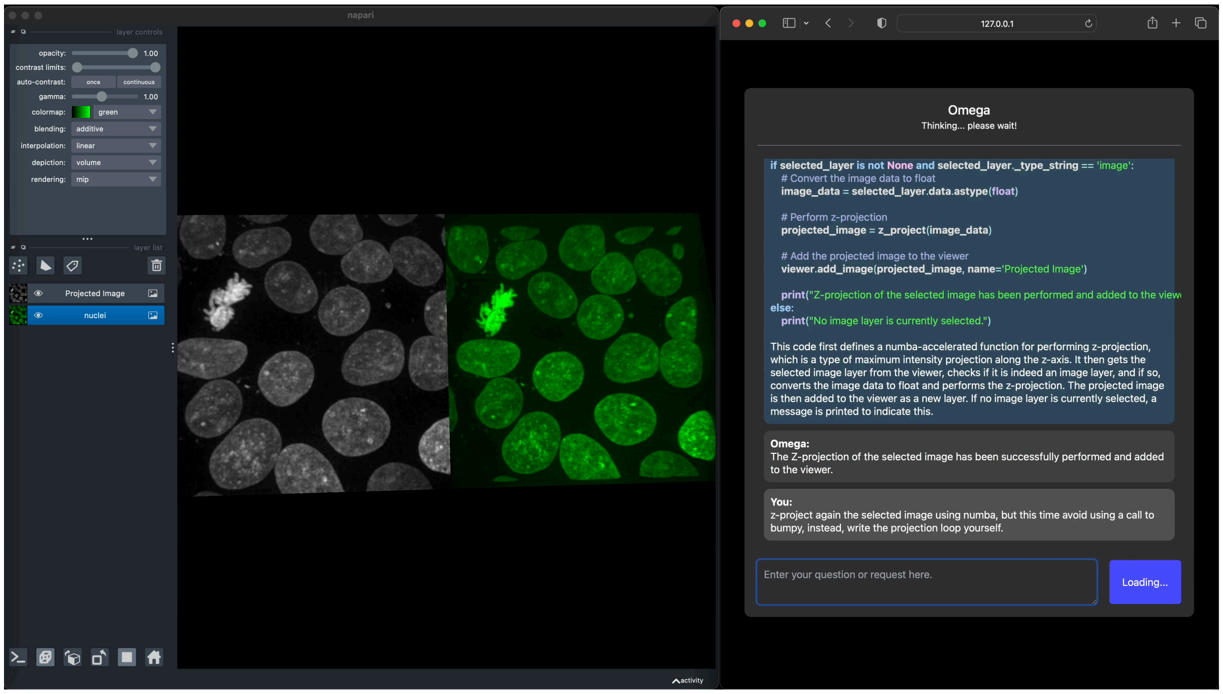
Supplementary Video 18. Omega knows how to use the scikit-image library for processing and analyzing images. This video showcases Omega's mastery of the scikit-image library and image processing. We asked Omega to segment an image with bright coins on a dark background, but we realized that the background was not uniform. To correct the background, we consulted with Omega and learned about different algorithms that could be used. Initially, we attempted to use the rolling-ball algorithm, but we encountered some issues due to Omega’s use of a white tophat filter instead of a black tophat filter. We then tried CLAHE (Contrast Limited Adaptive Histogram Equalization), which worked reasonably well, but perhaps we should have used larger tiles. The video has been sped up by a factor of 2.



```
opacity: 1.00
contrast limits:
auto-contrast: once continuous
gamma: 1.00
colormap: RGB
blending: translucent_no_de
interpolation: nearest
```

```
Load the video file
cap = cv2.VideoCapture('https://github.com/intel-iot-devkit/sample-
videos/blob/master/people-detection.mp4?raw=true')
Load the pre-trained model
hog = cv2.HOGDescriptor()
hog.setSVMDetector(cv2.HOGDescriptor_getDefaultPeopleDetector())
Loop through each frame of the video
while True: # Read the frame ret, frame = cap.read() if not ret: break
# Detect people in the frame
boxes, weights = hog.detectMultiScale(frame, winStride=(8, 8))

# Draw boxes around the detected people
for box in boxes:
    x, y, w, h = box
    cv2.rectangle(frame, (x, y), (x + w, y + h), (0, 255, 0), 2)

# Display the frame
cv2.imshow('frame', frame)
if cv2.waitKey(1) == ord('q'):
    break
Release the video file and close the window
cap.release() cv2.destroyAllWindows()
Please copy and paste this code into a new Python file and run it. Note that you will
need to have OpenCV installed in your Python environment for this code to work.'
```

Supplementary Video 19. Omega knows how to use OpenCV. In this video, we requested Omega to download an MP4 movie using the provided URL. The movie displays a hallway and people passing by – a commonly used video for testing person detection algorithms. We then asked Omega to utilize the OpenCV library to detect people in each movie frame and draw a bounding box around each detection. Omega complied with our request and displayed each frame and bounding boxes around each detected person. However, we observed two issues. Firstly, adding each 2D movie frame as individual napari image layers is impractical, resulting in many layers. Secondly, OpenCV's RGB channel ordering is incompatible, causing the napari viewer to display incorrect colors for each frame. The video has been sped up by a factor of 2.



```
opacity: 1.00
contrast limits:
auto-contrast: once continuous
gamma: 1.00
colormap: green
blending: additive
interpolation: linear
depiction: volume
rendering: mip
```

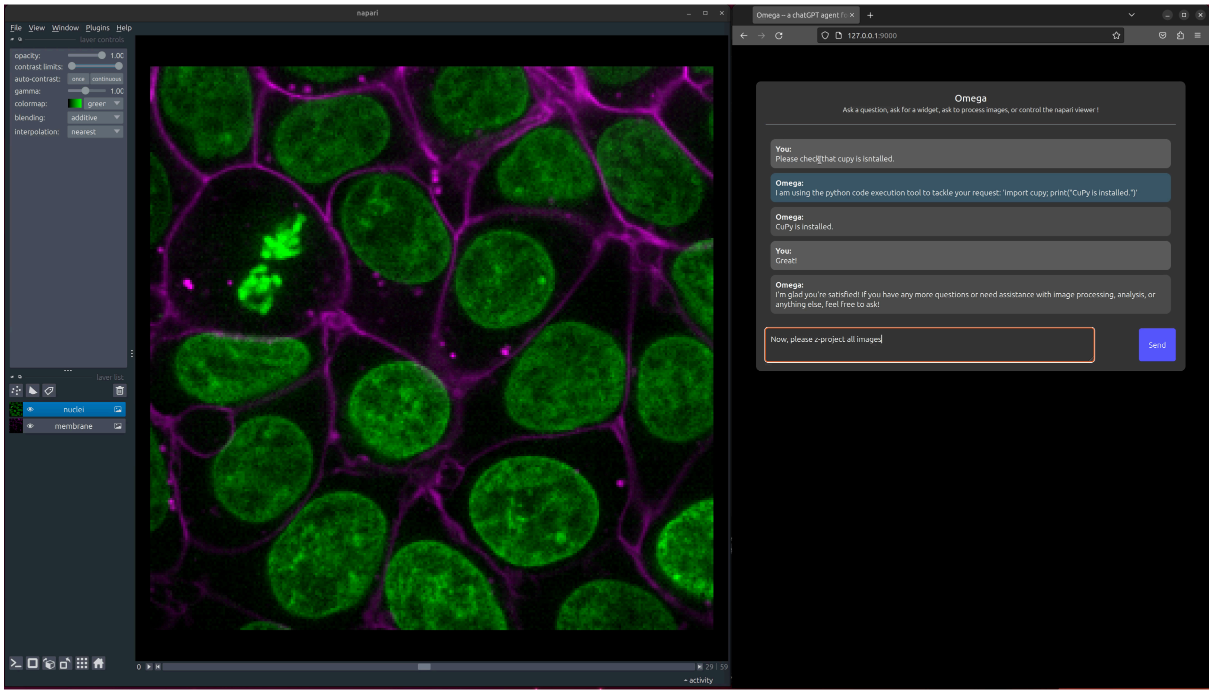
```
if selected_layer is not None and selected_layer._type_string == 'image':
    # Convert the image data to float
    image_data = selected_layer.data.astype(float)

    # Perform z-projection
    projected_image = z_project(image_data)

    # Add the projected image to the viewer
    viewer.add_image(projected_image, name='Projected Image')

    print("Z-projection of the selected image has been performed and added to the view
else:
    print("No image layer is currently selected.")
```

This code first defines a numba-accelerated function for performing z-projection, which is a type of maximum intensity projection along the z-axis. It then gets the selected image layer from the viewer, checks if it is indeed an image layer, and if so, converts the image data to float and performs the z-projection. The projected image is then added to the viewer as a new layer. If no image layer is currently selected, a message is printed to indicate this.

Omega:
The Z-projection of the selected image has been successfully performed and added to the viewer.

You:
z-project again the selected image using numba, but this time avoid using a call to bumpy, instead, write the projection loop yourself.

Supplementary Video 20. Omega knows how to use Numba. In the video, we asked Omega to perform a z-projection of a 3D image using the Numba library to speed up the code through just-in-time compilation. Although we did not specify the projection type, Omega used the reasonable choice of max projection and successfully computed it. However, during the process, Omega utilized the NumPy function np.max() in the just-in-time compiled function, defeating our purpose. We then requested Omega to refrain from using NumPy functions and instead write a z-projection loop. Omega completed the task, but this time, it opted for an average projection. We later explicitly asked Omega to perform a max projection. The video has been sped up by a factor of 2.



Omega
Ask a question, ask for a widget, ask to process images, or control the napari viewer !

You:
Please check that cupy is isntalled.

Omega:
I am using the python code execution tool to tackle your request: 'import cupy; print("CuPy is installed.")'

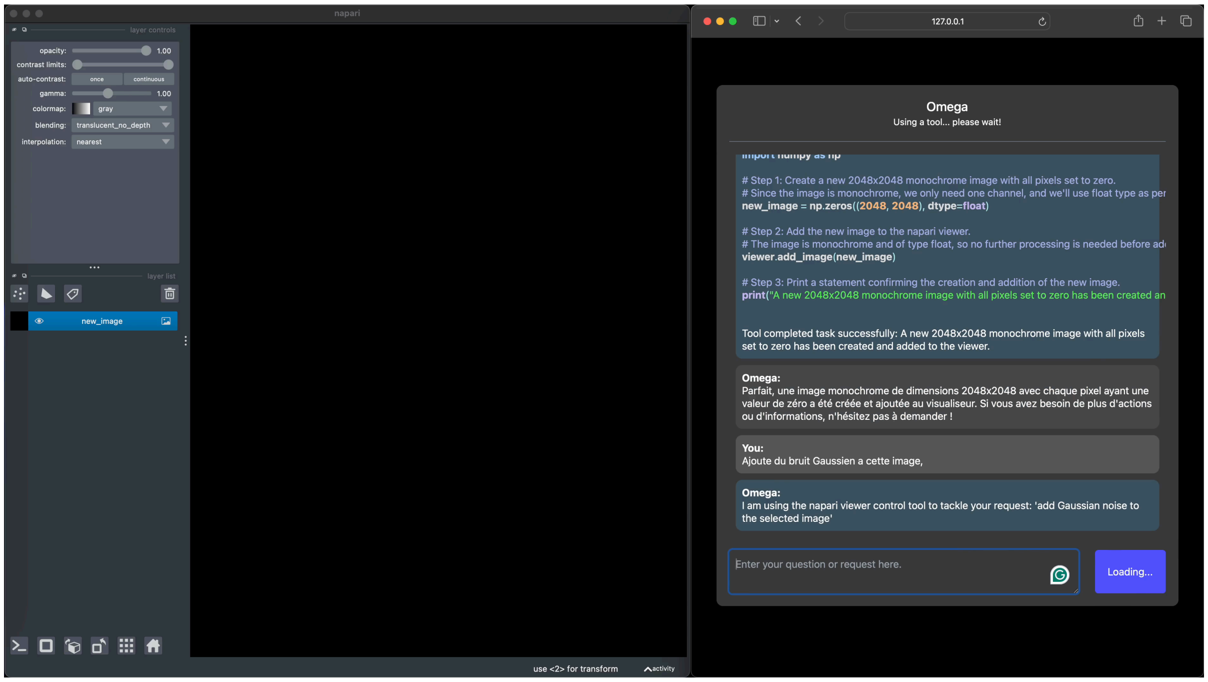
Omega:
CuPy is installed.

You:
Great!

Omega:
I'm glad you're satisfied! If you have any more questions or need assistance with image processing, analysis, or anything else, feel free to ask!

Now, please z-project all images

Supplementary Video 21. Omega knows how to use CuPy. This video presents Omega's proficiency in utilizing the GPU-accelerated CuPy library. Initially, we requested Omega to confirm the installation and functionality of CuPy. Subsequently, we instruct Omega to perform a z-projection of all images displayed in napari. The video has been sped up by a factor of 2.



Omega
Using a tool... please wait!

```
import numpy as np

# Step 1: Create a new 2048x2048 monochrome image with all pixels set to zero.
# Since the image is monochrome, we only need one channel, and we'll use float type as per
new_image = np.zeros((2048, 2048), dtype=float)

# Step 2: Add the new image to the napari viewer.
# The image is monochrome and of type float, so no further processing is needed before ad
viewer.add_image(new_image)

# Step 3: Print a statement confirming the creation and addition of the new image.
print("A new 2048x2048 monochrome image with all pixels set to zero has been created an
```

Tool completed task successfully: A new 2048x2048 monochrome image with all pixels set to zero has been created and added to the viewer.

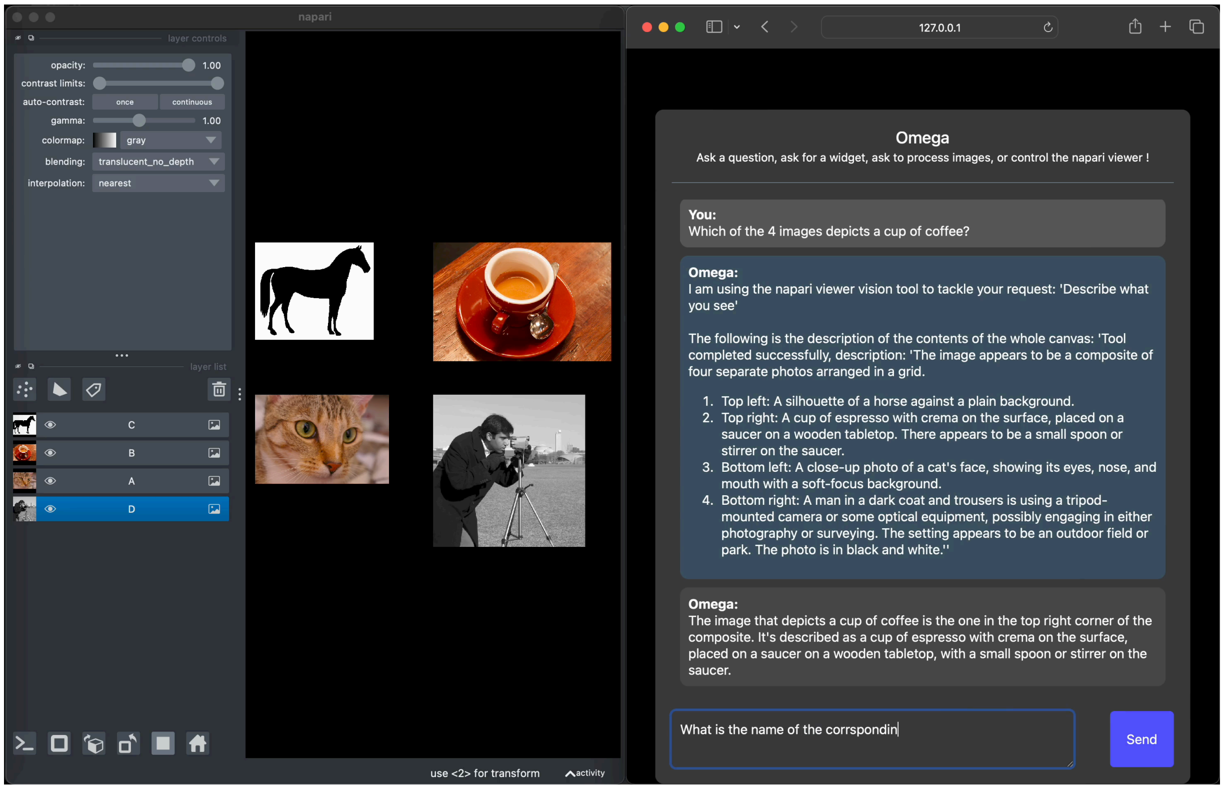
Omega:
Parfait, une image monochrome de dimensions 2048x2048 avec chaque pixel ayant une valeur de zéro a été créée et ajoutée au visualiseur. Si vous avez besoin de plus d'actions ou d'informations, n'hésitez pas à demander !

You:
Ajoute du bruit Gaussien a cette image,

Omega:
I am using the napari viewer control tool to tackle your request: 'add Gaussian noise to the selected image'

Enter your question or request here.

Supplementary Video 22. Omega can dialog in many different languages. In this video, we speak with Omega in French. This is possible because most LLMs (ChatGPT, Claude, and others) are naturally multilingual. Omega replies to the user in French, but the tools used still operate internally in English, as most of the prompt templates are written in that language. We have tested Omega in several languages, including Spanish, Italian, German, and even Chinese. This feature enhances accessibility to non-English speakers. The video has been sped up by a factor of 2.



opacity: 1.00
contrast limits:
auto-contrast: once continuous
gamma: 1.00
colormap: gray
blending: translucent_no_depth
interpolation: nearest

Omega

Ask a question, ask for a widget, ask to process images, or control the napari viewer !

You:
Which of the 4 images depicts a cup of coffee?

Omega:
I am using the napari viewer vision tool to tackle your request: 'Describe what you see'

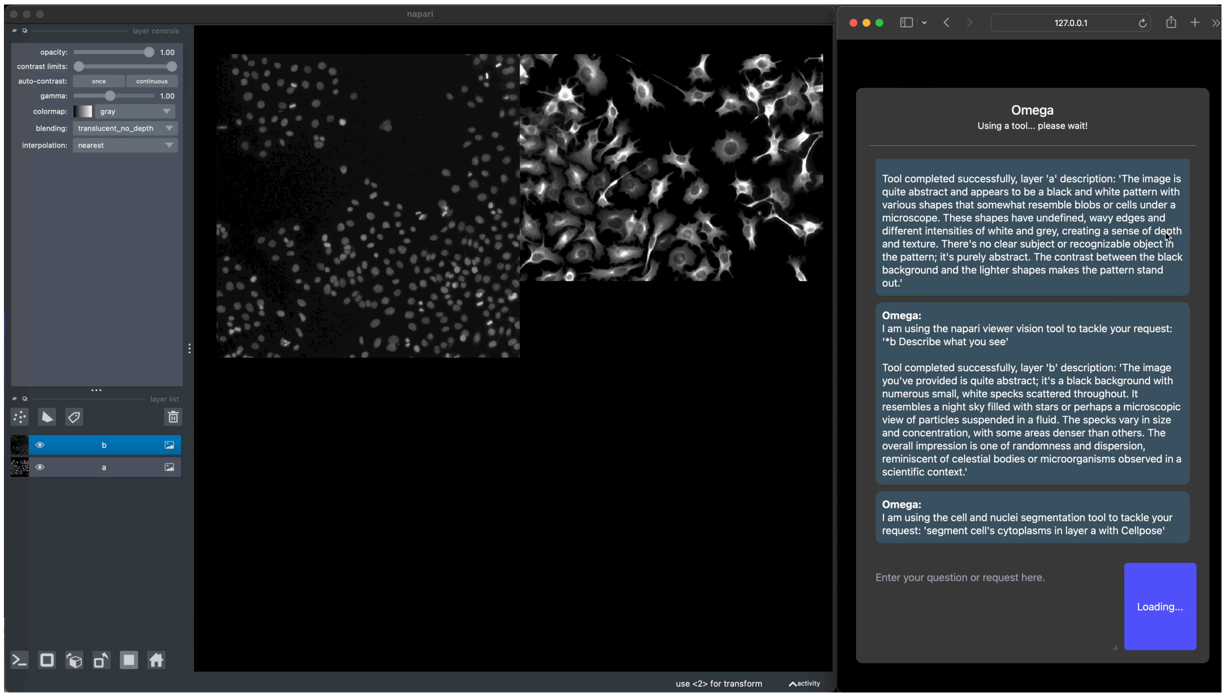
The following is the description of the contents of the whole canvas: 'Tool completed successfully, description: 'The image appears to be a composite of four separate photos arranged in a grid.

1. Top left: A silhouette of a horse against a plain background.
2. Top right: A cup of espresso with crema on the surface, placed on a saucer on a wooden tabletop. There appears to be a small spoon or stirrer on the saucer.
3. Bottom left: A close-up photo of a cat's face, showing its eyes, nose, and mouth with a soft-focus background.
4. Bottom right: A man in a dark coat and trousers is using a tripod-mounted camera or some optical equipment, possibly engaging in either photography or surveying. The setting appears to be an outdoor field or park. The photo is in black and white.''

Omega:
The image that depicts a cup of coffee is the one in the top right corner of the composite. It's described as a cup of espresso with crema on the surface, placed on a saucer on a wooden tabletop, with a small spoon or stirrer on the saucer.

What is the name of the corrspondin

Supplementary Video 23. **Omega can 'see'.** In this video, we test Omega's ability to see by giving it a visual puzzle to solve. We load 4 images (black & white horse, cup of coffee, cat face, camera test image), change their layer names to 'A', 'B', 'C', 'D', and then ask Omega to find which of the four images depicts a cup of coffee. Omega uses its 'napari viewer vision tool' to see the contents of the napari viewer, correctly describe each image, and identify the one depicting a cup of coffee as the one on the top right corner. We further ask for the name of the layer, and Omega uses again its vision tool to verify that layer 'B' holds that image. The final answer is the correct one: 'B'. The video has been sped up by a factor of 2.



opacity: 1.00
contrast limits:
auto-contrast: once continuous
gamma: 1.00
colormap: gray
blending: translucent_no_depth
interpolation: nearest

Omega

Using a tool... please wait!

Tool completed successfully, layer 'a' description: 'The image is quite abstract and appears to be a black and white pattern with various shapes that somewhat resemble blobs or cells under a microscope. These shapes have undefined, wavy edges and different intensities of white and grey, creating a sense of depth and texture. There's no clear subject or recognizable object in the pattern; it's purely abstract. The contrast between the black background and the lighter shapes makes the pattern stand out.'

Omega:
I am using the napari viewer vision tool to tackle your request: '*b Describe what you see'

Tool completed successfully, layer 'b' description: 'The image you've provided is quite abstract; it's a black background with numerous small, white specks scattered throughout. It resembles a night sky filled with stars or perhaps a microscopic view of particles suspended in a fluid. The specks vary in size and concentration, with some areas denser than others. The overall impression is one of randomness and dispersion, reminiscent of celestial bodies or microorganisms observed in a scientific context.'

Omega:
I am using the cell and nuclei segmentation tool to tackle your request: 'segment cell's cytoplasms in layer a with Cellpose'

Enter your question or request here.

Supplementary Video 24. **Omega decides how to best segment an image using vision.** In this video we present Omega with two 2D microscopy images: one with cytoplasm labeled (b) and another with nuclei labeled (a). We then ask Omega to decide how to best segment the biological structures present in each image by using the vision tool. Omega looks at the image contents, describe them, and correctly decides that Cellpose is best for layer b and StarDist is best for layer a.

Type	Prompts:	Dataset or Image to use:	Notes:
Filtering	Please make me a Gaussian blur widget with sigma parameter	E.g., the 'Human Mitosis' built-in napari sample dataset.	
	Please make me a widget for spatially variable Gaussian blur with a sigma parameter varying over z between two values: sigma_bottom and m sigma_top. The top corresponds to high-z values, and the bottom corresponds to low-z values. Sigma values should range between 0 (no blur) and 10.	3D image	
	Please make a widget that applies Butterworth filtering to a 2D grayscale image with a defined cut-off frequency parameter normalized between 0 (min. freq.) and 1 (max. freq.) and order (between 1 and 10).	2D grayscale image	
	Please make a Sobel edge filter widget that works on 3D grayscale images. The user can choose an isotropic kernel size of 3, 5, or 7. Make sure to optimize the code.	3D image	
	Create a sharpening filter widget for 2D monochrome images and expose relevant parameters. Ensure that the image's total brightness is preserved. Expose a parameter to control the strength of the sharpening.	2D image	
	Make a band-pass filter widget that keeps frequencies in a 2D or 3D grayscale image between two frequencies: f_min, f_max. The frequencies are provided in the normalized range [0f, 1f].		
Transforms	Please make a widget that can crop a 3D image by specifying the number of pixels to remove on all sides (x-, x+, y-, y+, z-, z+).	3D image	
	Please make a widget that can up- and down-scale a 3D image anisotropically along x, y, z. The type of interpolation for each axis can be set independently.	3D image	
	Please make a widget that takes a 2D RGB image and returns one of the following layers: either one for the hue, for the saturation, or luminance (user choice). The resulting layer values should range between 0f and 1f for luminance and saturation but be RGB (fully saturated, maximal luminance) for the hue.	2D RGB image	
	Please write a widget to convert a 2D image from RGB to grayscale. The weights used to project the color pixels to grayscale are configurable. Ensure weights sum to 1. Add a gamma parameter for each component R, G, and B before projection. Apply the inverse geometric average gamma transformation to the resulting gray-scale value.	2D RGB image	
	Please make a widget that computes the max projections of a 3D monochrome image along an arbitrary axis parametrized by Euler angles. Use the Fourier transform and the projection-slice theorem. Pad the input image to have the same dimension along all axes if necessary.	3D image	
	Make a widget that fuses two 2D images using the wavelet transform. Offer the choice of transform. Expose relevant parameters	Two 2D grayscale images.	You will need to install PyWavelets and restart napari for Omega to be able to find it. Omega will attempt to install

… (continued in Supp. File)

Supplementary Table 1. Example widgets. Here is a list of example widgets that can be reliably generated using Omega. See the full table in the corresponding Supp. File. These prompts can be modified, adjusted, and extended in many ways. If Omega can't make a functional plugin the first time, or if the result is not exactly what is asked for, being more explicit and asking Omega to 'try again' often works. Ideally, these widgets are made once and then can be reused by running the code in Omega's code editor (see Supp. Fig. 2).

Reproducibility test — each prompt is run 10 times, and the success rate is the ratio of successes over tries.

Supplementary Video	Prompts:	Dataset to preload:	Success rate:
Prompt 1	Segment the cell nuclei present in the selected layer. Once done, count the number of segmented nuclei and measure the average segment area.	Load the 'Human Mitosis' built-in napari sample dataset.	80%
Prompt 2	Convert the selected RGB image to grayscale using human-perception-based weights. Then, apply a light Gaussian blur of sigma 1.5 to the resulting grayscale image. Finally, create a new RGB image that visualizes the gradient of this grayscale image: the hue is proportional to the angle of the gradient vector at each pixel, and the luminance is proportional to the magnitude of the gradient.	Load the 'Astronaut' RGB image.	90%
Prompt 3	Please create a widget that takes a 2D image as input and splits it into k*k tiles, where k is a parameter in the range [4, 256]; snap this parameter to the closest common divisor of both x and y dimensions. For each tile, compute the entropy. The output should be an image of the same dimensions as the input, which is obtained by upscaling the image of tile entropies (one pixel per tile) using a parameterizable interpolation method. The tile size should be a parameter of the widget.	Load the camera image.	80%
Prompt 4	Please write a widget that color projects a 3D stack along the Z axis. The hue of the projected pixel is proportional to the depth of the voxel of max intensity, the luminance is proportional to that max intensity, and the saturation is proportional to the contrast between the max intensity and the average intensity.	Load the 'Cells (3D + 2C)' built-in napari sample dataset. Keep the nuclei channel	100%
Prompt 5	Make a widget to rotate the hue of an RGB image and adjust the gamma for the luminance and the gamma for saturation. The default for the angle and gamma values is 1.0. The range of the gamma values is: [0.01, 10].	Load the 'Astronaut' RGB image.	100%
Average			90%

All prompts were run with default Omega settings and model 'gpt-4-0125-preview'.

Supplementary Table 2. Reproducibility Analysis. We conducted a reproducibility analysis in which two complex multi-step prompts and three widget generation prompts were run ten times to assess reproducibility – a total of 50 runs. The results suggest that widget generation is more robust than complex multi-step tasks. We observed a 90% reproducibility rate (ratio of successful attempts versus all ten attempts). Each run was run independently with a blank conversation history. In general, our observation is that if Omega fails to follow instructions, it often suffices to ask Omega to "try again".

Omega Paper Prompts for Supplementary Videos

Supplementary Video	Prompts:	Dataset to load and notes:
Supp. Video 1	Segment the cell nuclei present in the selected layer.	Load the ‘Human Mitosis’ built-in napari sample dataset.
	Count the number of segmented nuclei.	
	Write a csv file on the desktop that lists all segmented nuclei, together with their area, and coordinates. Sort the rows by decreasing area. Open the file when done.	
Supp. Video 2	Segment the cell nuclei on the selected 3D image.	Load the ‘Cells (3D + 2C)’ built-in napari sample dataset. Keep the nuclei channel
	Count the number of segments	
Supp. Video 3	Make a step-by-step plan to segment nuclei in a 2D image. The nuclei are brighter than the background.	Load the ‘Human Mitosis’ built-in napari sample dataset.
	Let’s apply step 3	
	For step 4, let’s do two erosions	
	Would it not be better to apply erosions first on the grey-level image instead?	
	Let’s do step 3 with just one opening operation.	
	Let’s do one step of erosion on the selected image please.	
	Let’s go back to the steps, apply 4 and 5.	
Supp. Video 4	Segment the nuclei on the selected 2D image.	Load the ‘Human Mitosis’ built-in napari sample dataset.
	Make a widget that takes a labels layer and returns a new labels layer but filtered. Only labels within a provided range of areas (min_area, max_area) are kept in this new layer.	
Supp. Video 5	Please write a widget that color projects a 3D stack along the Z axis. The hue of the projected pixel is proportional to the depth of the voxel of max intensity, the luminance is proportional to that max intensity, and the saturation is proportional to the contrast between the max intensity and the average intensity.	Load the ‘Cells (3D + 2C)’ built-in napari sample dataset. Keep the nuclei channel
Supp. Video 6	Please make a widget that returns the FFT spectrum of a 2D image as the absolute logarithm of the Fourier transform magnitude. Ensure that the DC component is at the center of the image. Use reflection padding and apodization to reduce artifacts due to discontinuities at the image borders.	Load the camera image and load the ‘Cells (3D + 2C)’ built-in napari sample dataset. Keep the nuclei channel. This way, one can test the original 2D widget and the one modified by the AI tool in the editor.
Supp. Video 7	N/A	
Supp. Video 8	Hello, who are you?	
	Please create a 2D image of dimensions 2000x2000 filled with random Gaussian noise of zero mean and sigma=100	
Supp. Video 9	Segment image on first layer using the SLIC superpixel segmentation algorithm.	Load the ‘coffee cup’ built-in napari sample image.

… (continued in Supp. File)

Supplementary Table 3. Prompt Table. This table (see the full table in the corresponding Supp. File) lists all prompts used for