# Pyxel 1.0: an open source Python framework for detector and end-to-end instrument simulation

# Pyxel 1.0: an open source Python framework for detector and end-to-end instrument simulation

**Matej Arko,[a] Thibaut Prod'homme,[a,*] Frédéric Lemmel,[a] Benoit Serra,[b] Elizabeth George,[b] Bradley Kelman,[c] Thibault Pichon,[d] Enrico Biancalani,[e] and James Gilbert[f]**

[a]European Space Agency, ESTEC, Noordwijk, The Netherlands
[b]European Southern Observatory, Garching bei München, Germany
[c]The Open University, Milton Keynes, United Kingdom
[d]Université Paris-Saclay, CEA, CNRS, AIM, Gif-sur-Yvette, France
[e]Leiden University, Leiden, The Netherlands
[f]The Australian National University, Canberra, Australian Capital Territory, Australia

**Abstract.** Detector modeling is becoming more and more critical for the development of new instruments in scientific space missions and ground-based experiments. Modeling tools are often developed from scratch by each individual project and not necessarily shared for reuse by a wider community. To foster knowledge transfer, reusability, and reliability in the instrumentation community, we developed Pyxel, a framework for the simulation of scientific detectors and instruments. Pyxel is an open-source and collaborative project, based on Python, developed as an easy-to-use tool that can host and pipeline any kind of detector effect model. Recently, Pyxel has achieved a new milestone: the public release and launch of version 1.0, which simplified third-party contributions and improved ease of use even further. Since its launch, Pyxel has been experiencing a growing user community and is being used to simulate a variety of detectors. We give a tour of Pyxel's version 1.0 changes and new features, including a new interface, parallel computing, and new detectors and models. We continue with an example of using Pyxel as a tool for model optimization and calibration. Finally, we describe an example of how Pyxel and its features can be used to develop a full-scale end-to-end instrument simulator. © *2022 Society of Photo-Optical Instrumentation Engineers (SPIE)* [DOI: 10.1117/1.JATIS.8.4.048002]

**Keywords:** detector simulation; instrument simulation; imaging sensors; modelling; Python; charge-coupled device; complementary metal oxide semiconductor; calibration.

Paper 22084G received Sep. 2, 2022; accepted for publication Nov. 29, 2022; published online Dec. 24, 2022.

## 1 Introduction

Pyxel, the collaborative detection simulation framework, was born in 2017, when a team of physicists, detector experts, and software engineers from ESA's Science Payload Validation section decided to develop a Python-based detector simulation framework capable of hosting and pipelining any detector effect models.

ESA's Science payload validation section hosts a laboratory for detector characterisation. The laboratory is used to determine the performance of detectors in a representative environment (temperature, operating point, and radiation), validate the performance results obtained in other facilities (laboratories and companies), and carry out very specific experimental tests tailored to mission needs (e.g., scene[1] and subpixel spot projection,[2] irradiation at operating temperature,[3–5] persistence,[6] and electromagnetic compatibility). The interpretation of test results as well as the transfer of knowledge from lab to mission performance very often requires modeling and simulations. Several detector effects models were born that way: charge transfer inefficiency in charge-coupled devices (CCDs),[7,8] brighter fatter effect,[9,10] interpixel capacitance (IPC),[11,12] and persistence.[13] Similar models had already been developed by the community (e.g., Refs. 7, 8, 14,

*Address all correspondence to Thibaut Prod'homme, thibaut.prodhomme@esa.int and 15), sometimes even at ESA. At the same time, Python became more and more popular for simulations, data processing, and analysis among the lab team members and in the science community as well. Due to its simplicity, flexibility, object-oriented capabilities, and an active community, it was chosen as the main programming language. The idea of developing a Python framework that could host and pipeline existing models from different contributors started to take shape.

After several brainstorming sessions for a first conceptual architecture, the Pyxel team organized a survey internal to ESA among potential users (detector specialists, instrument and payload system engineers, optical engineers, etc.). From the survey, a first set of requirements were derived (summarized in the following). The positive feedback received fueled our motivation to develop Pyxel and a prototype was already on the table by June 2018 when it was presented at the SPIE astronomical telescopes and instrumentation conference in Austin, Texas.[16]

In 2019, a beta version was released and welcomed beta testers on the Pyxel Gitlab repository.[17] At that time, members of the European Southern Observatory detector group as well as several other members from the larger astronomy instrumentation community joined the Pyxel collaboration and helped us develop the tool further toward v1.0. Since 2020, the tool is capable of simulating CCDs, CIS (complementary metal oxide semiconductor image sensors or CMOS image sensors), hybridized mercury cadmium telluride (MCT) detector, and microwave kinetic inductance detectors (MKIDs) and operates in several modes including a model calibration mode, which can make use of laboratory test data to extract model parameters.

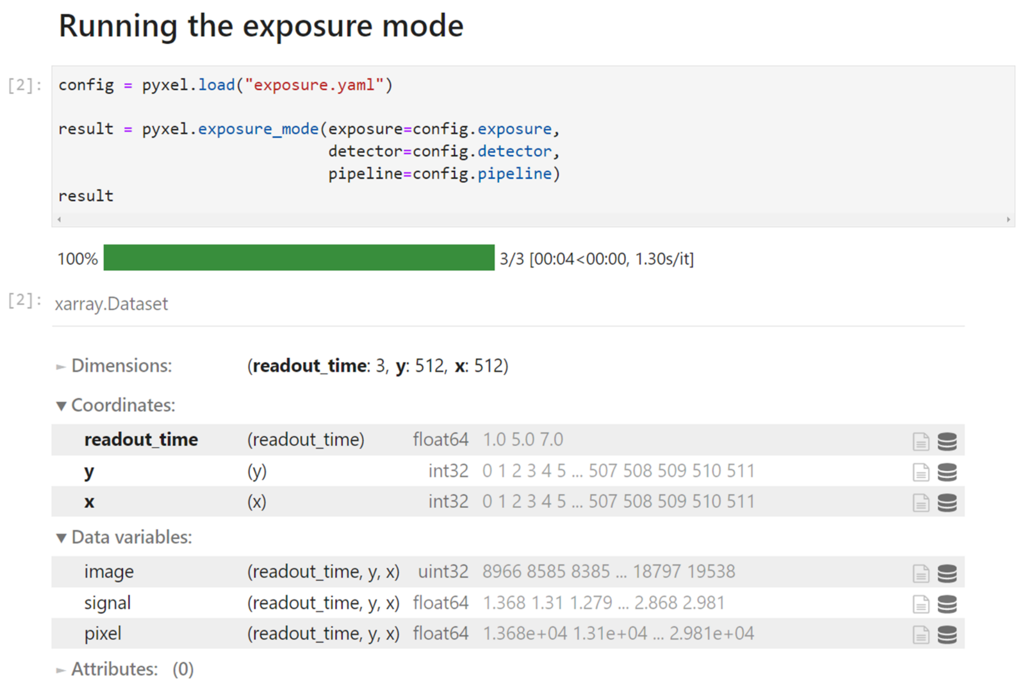
2021 has been a key year in the development of Pyxel toward v1.0, including milestones, such as reaching almost a hundred users in the Pyxel Gitlab repository,[17] the public release of Pyxel under the permissive MIT license and the release of tutorials explaining Pyxel's ins and outs: how to install it, how to use it, how to add a model, etc. The year culminated with the organization of a detector modeling workshop DeMo 2021 gathering several hundreds of detector and instrument simulation enthusiasts from the instrumentation community and demonstrating the potential of such a tool to foster knowledge transfer. Following the public release of Pyxel, version 1.0 was released in February 2022 and a release event was held. Since then, Pyxel is experiencing a rise in community engagement and has been even used for education purposes.

In this paper, we give an overview of the features and differences in the release 1.0. We continue with an explanation of the current state of the Pyxel architecture, such as configuration files and the main elements. We present the newest addition to the example repository and generic configuration files for realistic detectors. We calibrate such a pipeline using measurement data of a known detector and build an end-to-end instrument simulator. Finally, we outline a roadmap toward version 2.0.

## 2 Version 1.0

Pyxel is available for installation from either the Python Package Index, a software repository for the Python programming language, or Conda, the open-source package management system, under the name pyxel-sim. Pyxel runs on all of the three major operating systems: Windows, Linux, and MacOS. Before the final release of version 1.0, all the included models were internally reviewed and refactored. A more complete documentation and more code validation tests were added during the process. Some of the models were also optimized for speed.

The most significant change for the users came in the form of reorganization and simplification of running modes: four running modes were replaced with three. Modes previously known as single and dynamic were merged into one mode now called exposure. As its name suggests, it simulates a single exposure with the possibility of multiple detector readouts. Both nondestructive and destructive readout modes are possible. The former parametric mode was renamed to observation, simulating multiple exposures while changing either detector or model parameters. Using the dask[18] library, the observation mode pipelines can be run in parallel for faster computation either locally, on a grid or in a cluster of computers. Except for minor improvements, no changes were made to calibration mode.



Running the exposure mode

```
[2]: config = pyxel.load("exposure.yaml")

     result = pyxel.exposure_mode(exposure=config.exposure,
                                  detector=config.detector,
                                  pipeline=config.pipeline)
     result
```

```
100% ████████████████████ 3/3 [00:04<00:00, 1.30s/it]
```

```
[2]: xarray.Dataset

► Dimensions:      (readout_time: 3, y: 512, x: 512)
▼ Coordinates:
   readout_time    (readout_time)       float64  1.0 5.0 7.0
   y               (y)                  int32    0 1 2 3 4 5 ... 507 508 509 510 511
   x               (x)                  int32    0 1 2 3 4 5 ... 507 508 509 510 511
▼ Data variables:
   image           (readout_time, y, x) uint32   8966 8585 8385 ... 18797 19538
   signal          (readout_time, y, x) float64  1.368 1.31 1.279 ... 2.868 2.981
   pixel           (readout_time, y, x) float64  1.368e+04 1.31e+04 ... 2.981e+04
► Attributes:   (0)
```

**Fig. 1** A snapshot of the Jupyter notebook interface after simulating an exposure with 3 readout times, using the configuration file as shown in Fig. 2. The result is stored in a multidimensional dataset, in this case, it has three dimensions: the readout time, and the sensor coordinates *x* and *y* (essentially a collection of frames for different readouts).

Another major simplification in the new release is the standardization of the outputs. Results for all three running modes have the form of multidimensional datasets, provided by the Python library xarray,[19] which introduces labels in the form of dimensions, coordinates, and attributes on top of raw multidimensional arrays. Datasets can store detector data at different readout times in the case of exposure mode, detector data at a different model, or detector parameters in the case of observation mode, and simulated data and calibrated model parameters in the case of calibration mode. This new feature means a gain in efficiency for the user; tools developed to process Pyxel output data for a given mode can easily be reused or adapted now for another mode.

Two new detector types were implemented with the versions 1.0 and 1.1: MKIDs and avalanche photo diode (APD) arrays. Consequently, new models specific to the implemented detectors were added, such as a dead time filter for MKIDs and APD gain and readout noise for APDs. Some of the other models included in the new version are algorithm for charge transfer inefficiency correction,[8] temperature-dependent dark current model,[20] thermal (kTC) readout noise, successive-approximation-register analog-to-digital converter noise, and nonlinearity models.[21]

A significant portion of changes is aimed at an improved user experience. The command line interface was renewed, using the library click.[22] New functions for interactive plotting of results were added to be used in the Jupyter notebook interface,[23] see an example of running Pyxel simulation in Jupyter notebook interface in Fig. 1. Multiple examples, tutorials, and how-to guides in the form of Jupyter notebooks were added to a separate gitlab repository called Pyxel Data.[24] Tutorials are the starting point for any new user trying out the framework. The documentation[25] got a fresh look and was filled with more information about the models, architecture, and contribution guidelines. It is now easier than ever to develop new models and contribute to the software.

## 3 Architecture Overview

The main user entry point for any Pyxel simulation is a configuration file. With the configuration file, the three main elements in Pyxel are defined: the running mode, the detector, and the detection pipeline. After summarizing the requirements that have guided the development of

Pyxel's architecture, we give a brief overview of the main working principles behind Pyxel and the state of its architecture as of version 1.0.

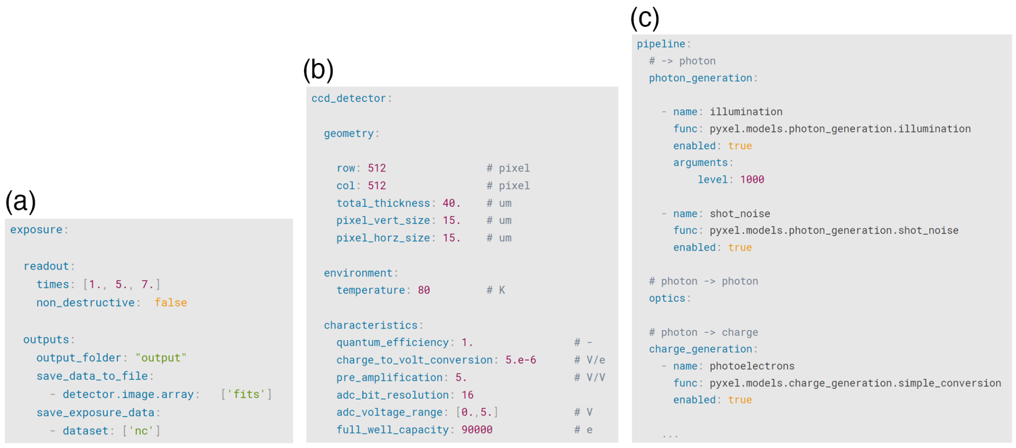
## 3.1 *Requirements*

At the origin of Pyxel, a user survey was performed to collect expectations among future potential users at ESA. From this exercise, a list of requirements has been derived to guide the design process. The following list provides a summary of the main driving requirements.

- Pyxel shall be implemented primarily in Python and shall enable collaborative and open-source development.
- Pyxel shall be user-friendly, well-documented, and runnable (with and without installation) from the main operating systems or online.
- Pyxel shall make maximum reuse of existing and widely used (python) libraries.
- Pyxel shall implement unit, functional, and integration tests.
- Pyxel shall enable model parametric analysis and facilitate model validation against test data.
- Pyxel shall be flexible enough to simulate any kind of (imaging) detectors and allow for both fast less accurate and slow more accurate simulations.
- Pyxel shall enable rapid implementation of detector models (by users external to main development team) and enable the pipelining of those models.

## 3.2 *Configuration File*

The configuration file is based on YAML, a user-friendly data serialization language.[26] The file consists of three separate parts, each representing a class in Pyxel architecture. They define the running mode, the detector properties, and the pipeline—the models the user wants to apply. When the YAML configuration file is loaded, the nested dictionaries, lists, numbers, and strings are used to directly initialize the Pyxel classes, see examples of the three main parts of the configuration file in Fig. 2. Despite the configuration file being human-readable and easy to understand, it is still possible to make mistakes that result in errors during the simulation. Therefore, a configuration file validation process based on JavaScript Object Notation (JSON) schema[27] is currently in development, which will further improve the user experience.


(a)
```yaml
exposure:

  readout:
    times: [1., 5., 7.]
    non_destructive:  false

  outputs:
    output_folder: "output"
    save_data_to_file:
      - detector.image.array:   ['fits']
    save_exposure_data:
      - dataset: ['nc']
```

(b)
```yaml
ccd_detector:

  geometry:

    row: 512                # pixel
    col: 512                # pixel
    total_thickness: 40.    # um
    pixel_vert_size: 15.    # um
    pixel_horz_size: 15.    # um

  environment:
    temperature: 80         # K

  characteristics:
    quantum_efficiency: 1.                # -
    charge_to_volt_conversion: 5.e-6      # V/e
    pre_amplification: 5.                 # V/V
    adc_bit_resolution: 16
    adc_voltage_range: [0.,5.]            # V
    full_well_capacity: 90000             # e
```

(c)
```yaml
pipeline:
  # -> photon
  photon_generation:

    - name: illumination
      func: pyxel.models.photon_generation.illumination
      enabled: true
      arguments:
          level: 1000

    - name: shot_noise
      func: pyxel.models.photon_generation.shot_noise
      enabled: true

  # photon -> photon
  optics:

  # photon -> charge
  charge_generation:
    - name: photoelectrons
      func: pyxel.models.charge_generation.simple_conversion
      enabled: true

    ...
```

**Fig. 2** Examples of a Pyxel configuration file: definitions of (a) running mode, (b) detector, and (c) simulation pipeline are the three main parts of the human-readable YAML configuration file in Pyxel. These three parts represent classes inside the Pyxel architecture.

## 3.3 *Detector*

The detector object is the main input of the detection pipeline. Therefore, it is a container for all the data that the models need access to inside the pipeline. First are the properties of the detector itself, falling in either of the following categories: geometry, characteristics, and environment, as shown in Fig. 3. Those are the properties that are used by more than one model. They do not change during a pipeline run and can vary depending on the detector used. Another category of detector properties, material, was temporarily removed in version 1.0 due to nonuse. Additionally, since version 1.0, properties were given user-friendly names with whole words instead of symbols and abbreviations. The detector also holds data buckets storing the simulated data, such as input photon distribution (photons), number of charge carriers generated (carrier type), signal variation in pixels (voltage and phase), and digitised image value (ADU). The data buckets are modified by the models in the pipeline and the state of the output detector at the end of the pipeline is changed. Finally, the detector tracks absolute time during an exposure and time since the last readout. Currently, implemented detectors inside Pyxel are CCD, hybrid and monolithic CMOS, MKID, and APD, with a wide selection of various detector effect models.

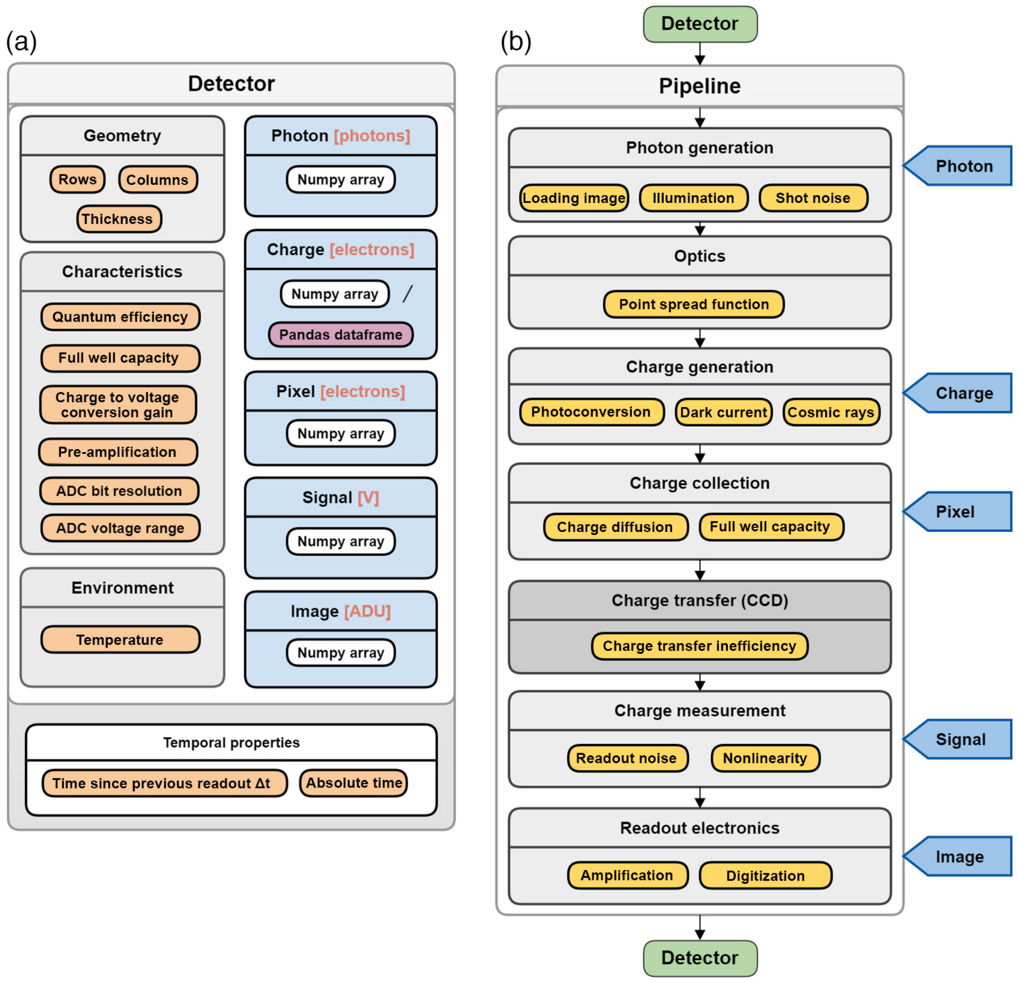


**Fig. 3** Two of the main objects in Pyxel's architecture are: (a) the detector object and (b) the pipeline. The detector contains detector properties, simulated data and temporal properties. It is the main input to the pipeline. Models inside the pipeline are grouped into model groups representing the working principles of the detection process; they modify the data inside the detector during simulation. The architecture shown above is valid in the case of CCD, CMOS, or APD detectors and changes slightly in the case of the MKID detector.

## 3.4 *Pipeline*

During simulation, the detector object passes through the pipeline, where models are applied one after the other (in the order defined in the configuration file), modifying the data stored inside the detector. Each model describes a unique physical phenomenon. Those can be the conversion of photons to photoelectrons, various noise sources, cosmic rays, etc. Models inside the pipeline are grouped into model groups, resembling the working principles of the detectors. In the case of a CMOS, these are photon generation, optics, charge generation, charge collection, charge measurement, and readout electronics, as shown in Fig. 3. Users can change model parameters or enable/disable them by interacting with the configuration file. It is also possible to add any kind of new or already existing models to Pyxel using the model plug-in mechanism. The instructions for adding models and the full list of already available models can be found in the documentation[25] including detailed explanations, literature references when available, and examples of configuration.

## 3.5 *Running Modes*

As shown in Fig. 4, the three different running modes in version 1.0 are as follows.

- *Exposure*. Pipeline is run one or more times, depending on the number of configurable readout times and useful for simple simulations, quick checks, and simulating time-dependent effects. Exposure mode cannot be run in parallel.
- *Observation*. Multiple exposures looping over a range of model or detector parameters and useful for parameter sensitivity analysis, checks of parameter space, simulating variable astronomical sources, etc. Parallelization is possible in observation mode.
- *Calibration*. Optimization of model parameters so they reproduce target datasets and useful for calibrating models, optimizing instrument performance, or retrieving detector physical properties from measurements. The optimization algorithm and optimized figure of merit are configurable. The built-in optimization algorithms are advanced genetic algorithms based on the pygmo package:[28] ideal for wide/degenerate parameter space and nonlinear problems. It must be run in parallel since the number of pipelines that are run each time is very high.

All three modes of operation share the same format of outputs that are a multidimensional dataset.

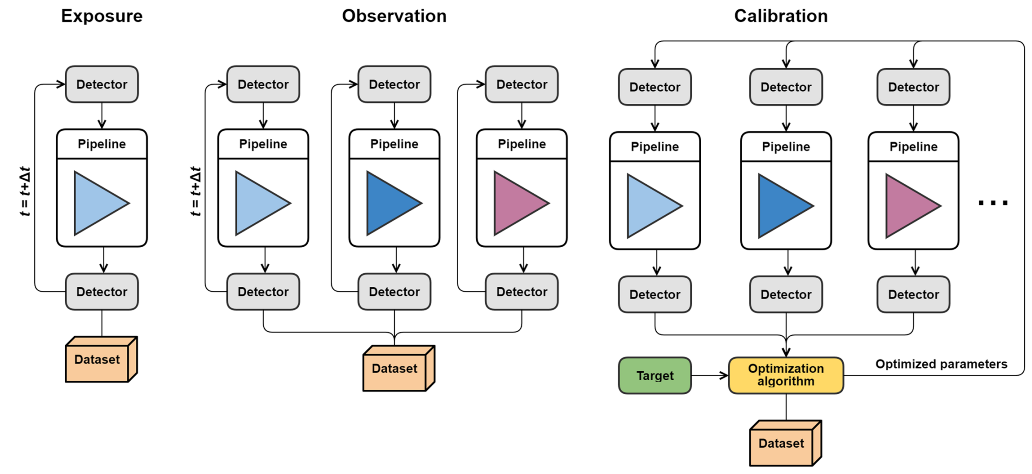


**Fig. 4** The three running modes of Pyxel: exposure, observation, and calibration. In all three cases, the results are stored in a multidimensional dataset with the same data format. Both observation and calibration modes have the possibility to run with dask on a computer cluster or a grid of computers.

## 4 Examples

In this section, we present the newest addition to the collection of example notebooks in the Pyxel Data repository:[24] generic configuration files with realistic model and detector parameters for different types of detectors. We show how the pipeline can be adjusted and calibrated for a specific detector. Due to the availability of measurement data and models, we choose the Teledyne H4RG infrared MCT-hybridized CMOS detector.[29] We show how the calibrated pipeline in Pyxel can be used in symbiosis with other simulation tools to create an end-to-end simulation tool with a full set of detector effect models. We use Scopesim,[30] a Pythonic astronomical instrument simulator, to simulate photon flux on the MICADO detector plane. MICADO,[31] a first-generation Extremely Large Telescope (ELT) instrument, has a detector plane consisting of nine Teledyne H4RG detectors, which makes it a good study case.

### 4.1 *Generic Detector Pipelines*

The Pyxel model library contains models for various types of detectors. Not all models can be used with all of the detector types and some specific models are only to be used with a single type of detector. Until the configuration file validation based on JSON schema is fully implemented, the detector type validation is only performed inside the models during a simulation. For this reason and to help new users and nonexperts, generic configuration file templates for different detectors have been included in the Pyxel Data example repository, together with corresponding Jupyter notebooks. They include detector properties and pipelines with detector-appropriate sets of models, prefilled with realistic model argument values. They provide a good starting point for simulations of specific detectors and later customization or iteration with detector engineers and experts. The generic pipelines are now available for the following types of detectors: generic CCD, generic CMOS, Teledyne HxRG, and APD array detector based on Leonardo's Saphira detector.[32]

### 4.2 *Model Calibration*

To calibrate the Pyxel generic H4RG pipeline for simulation of the MICADO instrument, we use test data from ESO of an engineering grade H4RG-15 detector. The measurements[33] were done on a detector for the MOONS project,[34] but the detectors used for both projects are the same, the only difference being operating temperature and antireflective coating. The operating temperature for the MICADO detectors will be 80 K. Some basic detector parameters include a pixel pitch of 15 $\mu$m, resolution of $4096 \times 4096$, and cutoff at 77 K of 2.48 $\mu$m.

In additon to detector temperature and geometry, another prerequisite for a realistic configuration file is a correct set of detector characteristics that gives a correct system gain. We use a measurement of photon transfer curve (PTC),[35] as a baseline for calibration and comparison between simulations and measurements. As shown in Fig. 5, the PTC is linear at mid-signal levels where it is mainly influenced by shot noise and moves away from a linear slope due to nonlinearity and saturation effects at higher signal levels. The variance starts to drop to 0 when the full well capacity is reached, at the point called Poissonian full well. The system gain in $\mathrm{e}^-/\mathrm{ADU}$ can be extracted with linear regression as the inverse of the PTC slope in the region where the noise behaves in a Poissonian fashion. In our case, the measured system gain is 2.28 $\mathrm{e}^-/\mathrm{ADU}$.

The known values for the detector are transimpedance gain (at the starvation level) of 5.93 $\mu\mathrm{V}/\mathrm{e}^-$, preamplifier gain of 6, and the gain of the analog to digital conversion of 78.125 $\mu\mathrm{V}/\mathrm{ADU}$. From the latter, we can extract the ADC voltage range of 5.12 V for a 16-bit converter, which is used in the configuration file. The actual ADC range is $\pm 4.096$ V, but due to a gain of 1.6 in the ADC differential driver amplifier, the effective swing at the input is $\pm 2.56$ V. Using the transimpedance gain and charge of an electron, we can estimate the total integrating capacitance at full depletion as $1.602 \times 10^{-19}\ \mathrm{C}/5.93\ \mu\mathrm{V} = 27$ fF. System gain can be calculated as

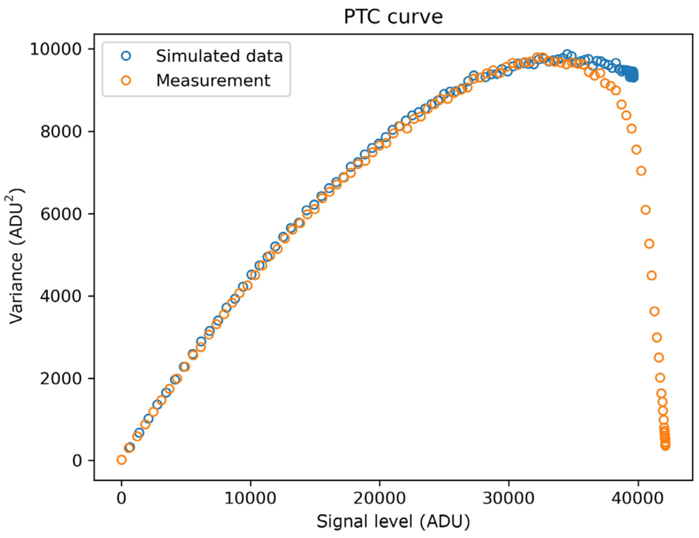


**Fig. 5** Comparison between the measured PTC and the simulated one. Due to effects of nonlinearity, at high signal values the noise stops behaving in a Poissonian fashion and the variance drops to 0. The analytical model of nonlinearity that does not simulate saturation fails to match the measurement at high signal values but has good overlap at lower signal values. The notebook to recreate the PTC curve is available in the /jatis folder on the Pyxel Data git repository.

$$\frac{78.125\ \mu\mathrm{V/ADU}}{6 \times 5.93\ \mu\mathrm{V/e^-}} = 2.196\ \mathrm{e^-/ADU}.$$

The theoretical value differs slightly from the measured one. The first reason is that the reported transimpedance gain is quoted at starvation level, where depletion is at the maximum, which is not the case when fitting the curve over a range of signal levels. The second effect is the IPC,[11,12] which exists due to an electric field between adjacent collection nodes, causing a spread of signal to adjacent pixels. The effect can be modeled as an additional point spread function, a convolution with an IPC kernel, which is extracted from measurements by observing an image of a hot pixel. The model in Pyxel convolves the image with a $3 \times 3$ IPC kernel. Simulating the PTC curve, we are able to recreate a system gain of 2.20 $\mathrm{e^-/ADU}$ without IPC and 2.34 $\mathrm{e^-/ADU}$ in case IPC model is applied. Both numbers are the result of a linear regression of the PTC in the region dominated by shot noise.

The second objective is recreating the effect of nonlinearity observed in the PTC. In a simplified analytical detector nonlinearity model,[21] which assumes that the detector is working far from saturation, the current flowing in the diode is restricted to a photonic current:

$$\frac{dV}{dt} = \frac{-I_{\mathrm{ph}}}{C}. \tag{1}$$

The integrating capacitance can be written as a sum of fixed capacitance $C_f$ in the readout integrated circuit and diode capacitance:

$$C = C_f + \frac{C_0}{1 - \frac{V}{V_{\mathrm{bi}}}}. \tag{2}$$

The diode capacitance at 0 bias $C_0$ is[36]

$$C_0 = A\sqrt{\frac{e\epsilon\epsilon_0}{2V_{\mathrm{bi}}}\left(\frac{1}{N_a} + \frac{1}{N_d}\right)}, \tag{3}$$

where $A$ is the area of the circularly shaped diode, $e$ is the electron charge, $\epsilon$ is the dielectric constant of the material, and $N_a$ and $N_d$ are the acceptor and donor concentrations. $V_{\mathrm{bi}}$ is the

**Table 1** Final nonlinearity model parameters as estimated with Pyxel's calibration mode (not provided by the manufacturer). The values are a result of calibrating the model parameters using lab data (PTC).

| Parameter | Value |
|---|---|
| Donor density $N_d$ | $2.9 \times 10^{15}$ (atoms/cm$^3$) |
| Diode diameter $d$ | 10.2 ($\mu$m) |
| Fixed capacitance $C_f$ | 6.8 (fF) |

built-in diode potential and is a function of $N_a$, $N_d$, temperature and intrinsic carrier concentration. By inserting Eq. (2) into Eq. (1) and integrating, one can express voltage on the detector after exposure as a solution of a quadratic equation. Different to the nonlinearity model presented in Ref. 37, this model takes into account the additional fixed capacitance and simulates classical nonlinearity as well as the gain nonlinearity[38]. Without the additional fixed capacitance, the resulting node capacitance would be lower than what is observed in the standard H4RG detector. Still the described model is not a complete physical model and does not simulate the saturation of the detector. Additionally, it assumes a planar geometry of the diode instead of a cylindrical.

We use Pyxel's calibration mode to estimate the physical parameters of the nonlinearity model directly using the PTC measurement, namely: the donor density $N_d$, the diode diameter $d$ giving the diode area, and the fixed capacitance $C_f$. Since H4RG is a p-on-n technology, the acceptor density has very little influence on the result and is fixed to $10^{18}$ atoms/cm$^3$. The detector voltage bias is 0.25 V. The target data for calibration are frames with known variance at different signal levels, extracted from the PTC measurement. We perform the calibration using library pygmo.[28] Pygmo is a Python library for massively parallel optimization, offering a wide range of bioinspired genetic and optimization algorithms. We use the built-in genetic optimization self-adaptive differential evolution. The final parameters, seen in Table 1, are the result of running the algorithm on a population size of 30 for 200 generations on a local cluster with 10 workers. Similar parameters have been used in the IPC model in Ref. 12 and the donor concentration result corresponds to the typical values reported in Ref. 39. The diode diameter is within the size of the pixel and fixed capacitance within the total integrating capacitance of 27 fF.

With the final parameters, a PTC curve can be generated using the observation mode by iterating over the input photon signal level. A comparison between the measurement and the generated PTC with the calibrated pipeline is shown in Fig. 5. The notebook to recreate the PTC curve is available in the /jatis folder on the Pyxel Data git repository. The models applied in the pipeline include the nonlinearity model with the arguments from Table 1 as well as the IPC model. As expected, the analytical nonlinearity model used in the pipeline is not able to fully replicate the data at high signal levels since it is not a complete physical model and does not simulate saturation. Overlap is much better in the region where the total noise is dominated by Poissonian shot noise. Therefore the configured pipeline can be used to reproduce detector images with high fidelity for the lower end of the dynamic range.

## 4.3 *Building an End-to-End Instrument Simulator*

Even though the main purpose of Pyxel so far has been the simulation of detector effects on a single detector, its flexible architecture also enables simulations on a much larger scale, such as end-to-end simulations of an entire instrument or an array of detectors. One way to achieve this is by making use of Pyxel's modular structure and combining it with already existing simulation frameworks. Pyxel can be easily implemented into another simulation tool as a library, adding several detector effect models that might be missing inside another simulation pipeline. Another approach is to wrap other simulation tools into models that can be used directly in the Pyxel pipeline. Here we give an example of the latter approach and show how Pyxel can be used together with ScopeSim,[30] a Pythonic astronomical instrument simulator. The final aim for this is a demonstration of how one can perform an end-to-end simulation of an instrument, in this case, MICADO on the ELT, by expanding the complexity of the Pyxel pipeline.

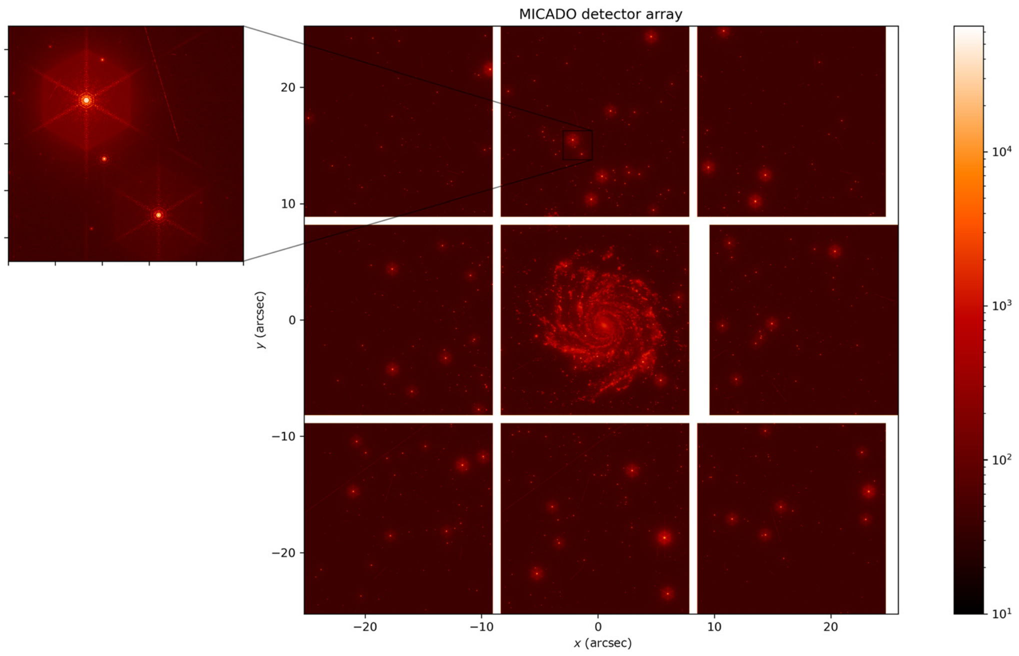


**Fig. 6** Simulation of a single observation of 200 s in H band with the MICADO instrument. ScopeSim framework with a dedicated MICADO instrument package is used to simulate the image plane—the photon distribution on the nine detectors. The observation mode in Pyxel is used to split the simulation in nine pipelines, adding detector effects to each of the detectors separately.

Inside the ScopeSim ecosystem, there are tools for the creation of astronomical on-sky targets, downloadable instrument data packages for the telescope (ELT), observing site (Armazones), the relay optics (MICADO), and tool Anisocado for simulation of point spread functions. We combine those into one model of photon generation, providing full photon flux of an astronomical source in the detector plane. For demonstration purposes, we use for the source a combination of a spiral galaxy with an extent of around 20 arcsec and a star cluster with $m = 10{,}000 M_{\odot}$ at a distance of 10 kpc. The MICADO detector plane consists of a $3 \times 3$ array of near-infrared Teledyne H4RG detectors, each with a resolution of $4096 \times 4096$. The map of the photon flux created by the Scopesim wrapper extents over all nine detectors. Exposure time was set to 200 s and the applied filter is in the H band. We use Pyxel's observation mode to split the simulation of detector effects into nine pipelines, separated for each of the predefined detector windows. This way each detector in the array is assigned an index and corresponds to a pipeline independent of other detectors. The pipeline used is the calibrated H4RG pipeline including, the calibrated nonlinearity model, IPC model, dark current, readout noise, and cosmic rays. The final image simulated using Pyxel can be seen in Fig. 6. It shows the full focal plane composed of nine H4RG detectors. Note that for computational speed purpose, we used a field-constant point spread function. A field-varying PSF is also possible inside the ScopeSim framework. In such case, using parallelization in the observation mode is necessary.

## 5 Conclusion

Detector and instrument simulation tools are crucial in all of the stages of the mission design. To avoid duplication of work, promote knowledge transfer, and provide reusable and reliable tool, we developed a collaborative tool Pyxel, an open source Python framework for detector and end-to-end instrument simulation. In this contribution, we presented the new features and gave an overview of architecture in the latest public release of Pyxel: version 1.0. One of the new features are generic configuration files for realistic detectors. Using such a generic configuration file for the case of the H4RG detector, we calibrated models using measurement data and ultimately used the calibrated pipeline to simulate the MICADO instrument focal plane comprising 9 H4RG sensors.

With a growing number of users, features and domain of applications, large validated set of models, a simplified architecture, and improved user experience, Pyxel has fulfilled the initial set of requirements and is showing to be a versatile and flexible tool that can simulate both single detectors as well as an entire instrument. Looking toward the future version 2.0, we would like to improve the speed of simulating large detectors, multichannel detectors, and arrays of detectors, either using multiprocessor computation or graphics processing unit. Other features often requested by users are a scene generator and multiwavelength simulations. For this reason, two new model groups called scene generation and photon collection are currently being implemented. We encourage users to join the Pyxel community, to contribute with their own models, and to collaboratively further expand the capability of the Pyxel pipeline.

## Acknowledgments

We would like to acknowledge ESO for providing detector measurement data and Naidu Bezawada for valuable advice and proofreading. We would also like to acknowledge Kieran Leschinski and the team of ScopeSim for support while trying to close the gap between the frameworks. Finally, we would like to thank Dávid Lucsányi, Hans Smit, and all past Pyxel developers that have massively contributed toward the release of Pyxel version 1.0.

## References

1. T. Prod'homme et al., "A smartphone-based arbitrary scene projector for detector testing and instrument performance evaluation," *Proc. SPIE* **11454**, 1145426 (2020).
2. P. Verhoeve et al., "Optical and dark characterization of the PLATO CCD at ESA," *Proc. SPIE* **9915**, 99150Z (2016).
3. T. Prod'homme et al., "Comparative study of cryogenic versus room-temperature proton irradiation of N-channel CCDs and subsequent annealing," *IEEE Trans. Nucl. Sci.* **66**, 134–139 (2019).
4. P. E. Crouzet et al., "Impact of proton radiation on the Ariel AIRS CH1 HAWAII-1RG MWIR detector," *Proc. SPIE* **11454**, 114540A (2020).
5. P. Verhoeve et al., "Proton induced damage after laboratory cold irradiation in CCD47-20 CCDs for CHEOPS," *Proc. SPIE* **12191**, 121910B (2022).
6. P.-E. Crouzet et al., "Comparison of persistence in spot versus flat field illumination and single pixel response on a Euclid HAWAII-2RG at ESTEC," *Proc. SPIE* **9915**, 99151E (2016).
7. A. Short et al., "An analytical model of radiation-induced charge transfer inefficiency for CCD detectors," *Mon. Not. R. Astron. Soc.* **430**(4), 3078–3085 (2013).
8. J. Kegerreis, R. Massey, and J. Nightingale, "Algorithm for charge transfer inefficiency (CTI) correction," 2022, https://github.com/jkeger/arctic.
9. P. Antilogus et al., "The brighter-fatter effect and pixel correlations in CCD sensors," *J. Instrum.* **9**, C03048 (2014).
10. A. A. Plazas et al., "Laboratory measurement of the brighter-fatter effect in an H2RG infrared detector," *Publ. Astron. Soc. Pac.* **130**, 065004 (2018).
11. A. Kannawadi et al., "The impact of interpixel capacitance in CMOS detectors on PSF shapes and implications for WFIRST," *Publ. Astron. Soc. Pac.* **128**, 095001 (2016).
12. K. Donlon et al., "Modeling of hybridized infrared arrays for characterization of interpixel capacitive coupling," *Opt. Eng.* **56**, 024103 (2017).
13. S. Tulloch, E. George, and ESO Detector Systems Group, "Predictive model of persistence in H2RG detectors," *J. Astron. Telesc. Instrum. Syst.* **5**, 036004 (2019).
14. J. Skottfelt et al., "C3TM: CEI CCD charge transfer model for radiation damage analysis and testing," *Proc. SPIE* **10709**, 1070918 (2018).
15. T. Prod'homme et al., "Electrode level Monte Carlo model of radiation damage effects on astronomical CCDs," *Mon. Not. R. Astron. Soc.* **414**, 2215–2228 (2011).
16. D. Lucsanyi et al., "Pyxel: a novel and multi-purpose Python-based framework for imaging detector simulation," *Proc. SPIE* **10709**, 107091A (2018).

17. Pyxel Development Team, "Pyxel: The collaborative detection simulation framework," *GitLab repository*, GitLab (2022).
18. M. Rocklin, "Dask: parallel computation with blocked algorithms and task scheduling," in *Proc. 14th Python in Science Conf.* , pp. 130–136 (2015).
19. S. Hoyer and J. Hamman, "xarray: N-D labeled arrays and datasets in Python," *J. Open Res. Softw.* **5**(1) (2017).
20. M. Konnik and J. Welsh, "High-level numerical simulations of noise in CCD and CMOS photosensors: review and tutorial," https://doi.org/10.48550/arXiv.1412.4031 (2014).
21. O. B. T. Pichon and T. Le Goff, "Pyxel: CMOS detector non-linearities," personal communication.
22. "Click: Python package for creating beautiful command line interfaces," https://click.palletsprojects.com/en/8.1.x/
23. T. Kluyver et al., "Jupyter Notebooks – a publishing format for reproducible computational workflows," *Positioning and Power in Academic Publishing: Players, Agents and Agenda*, F. Loizides and B. Schmidt, Ed., pp. 87–90, IOS Press (2016).
24. Pyxel Development Team, "Pyxel data: example notebook repository for pyxel," *GitLab repository*, GitLab (2022).
25. Pyxel Development Team, "Pyxel documentation," https://esa.gitlab.io/pyxel/doc/stable/index.html (2022).
26. O. Ben-Kiki, C. Evans, and I. döt Net, "YAML 1.2," http://yaml.org (2021).
27. F. Pezoa et al., "Foundations of JSON schema," in *Proc. 25th Int. Conf. World Wide Web*, pp. 263–273, International World Wide Web Conferences Steering Committee (2016)
28. F. Biscani and D. Izzo, "A parallel global multiobjective framework for optimization: pagmo," *J. Open Source Softw.* **5**(53), 2338 (2020).
29. R. Blank et al., "The HxRG Family of High Performance Image Sensors for Astronomy," in *Solar Polarization 6*, J. R. Kuhn et al., Eds., Astronomical Society of the Pacific Conference Series, Vol. **437**, pp. 383, Astronomical Society of the Pacific (2011).
30. K. Leschinski et al., "ScopeSim: a flexible general purpose astronomical instrument data simulation framework in Python," *Proc. SPIE* **11452**, 114521Z (2020).
31. R. W. Davies et al., "The MICADO first light imager for the ELT: overview, operation, simulation," *Proc. SPIE* **10702**, 107021S (2018).
32. I. Baker et al., "Linear-mode avalanche photodiode arrays in HgCdTe at Leonardo, UK: the current status," *Proc. SPIE* **10980**, 109800K (2019).
33. D. J. Ives et al., "Characterisation, performance and operational aspects of the H4RG-15 near infrared detectors for the MOONS instrument," *Proc. SPIE* **11454**, 114541N (2020).
34. M. Cirasuolo et al., "MOONS: the new multi-object spectrograph for the VLT," *Messenger* **180**, 10–17 (2020).
35. J. R. Janesick, "Photon transfer curve," in *Photon Transfer*, pp. 49–78, SPIE Press, Bellingham, Washington (2007).
36. S. Sze and K. K. Ng, Eds., "p-n Junctions," in *Physics of Semiconductor Devices* (2006).
37. A. Plazas et al., "Nonlinearity and pixel shifting effects in HXRG infrared detectors," *J. Instrum.* **12**, C04009 (2017).
38. N. Bezawada, D. Ives, and D. Atkinson, "Conversion gain non-linearity and its correction in hybridised near infrared detectors," *Proc. SPIE* **6690**, 669005 (2007).
39. C. Cervera et al., "Ultra-low dark current hgcdte detector in swir for space applications," *J. Electron. Mater.* **46**, 6142–6149 (2016).

**Matej Arko** received his master's degree in technical physics and photonics from the University of Ljubljana, Slovenia. During his two years at the European Space Agency as a young graduate trainee, he has been involved in the open source project Pyxel as the lead developer and maintainer of the simulation framework. Currently, he is working as an instrument scientist and performance engineer at SRON Netherlands Institute for Space Research, supporting the SpexONE instrument calibration and detector characterization for the ESA CO2M mission.

**Thibaut Prod'homme** received his engineering degree in materials and nanotechnologies from Rennes' National Institute for Applied Sciences, France, his master's degree in physics

from Rennes University, France, and his PhD in astronomy from Leiden University, The Netherlands, in 2011. He is a physicist at the European Space Agency (Payload Technology Validation Section, Future Mission Department, Science Directorate, ESTEC). Besides his role as payload manager for ESA's science missions in early phases, his work is mostly dedicated to technology development in the field of detectors: managing technology development activities in collaboration with the European industry, as well as carrying out both experimental and modeling works. He has been leading the Pyxel project since its very beginning.

**Frédéric Lemmel** received his MSc degree in computer engineering from Delft University of Technology. He has been working at the SCI/FIV Payload Technology and Validation Section at ESA. His main interests are software and hardware development on microcontrollers, FPGAs, and computers applied to CCD/CMOS detectors.

**Benoit Serra** is a detector engineer at ESO working on a first light instrument (METIS) for the ELT. He is mostly working with IR detectors, such as the HxRG and SAPHIRA. In the context of a collaboration between ESA and ESO, he has been involved in the development of Pyxel since the first beta release.

**Elizabeth George** received her PhD in physics from the UC Berkeley and is now a detector engineer at the European Southern Observatory. She is currently working on optical and infrared detector systems, including modeling and characterization, for the very and extremely large telescopes.

**Bradley Kelman** is a PhD student at the Open University studying the correction of charge transfer inefficiency (CTI) through the use of the Pyxel framework and the ArCTIC CTI model.

**Thibault Pichon** received his engineering degree in physics from Grenoble INP-Phelma in 2017 and his PhD from Université Paris-Saclay in 2020. During his PhD, he has been working on radiation effects in IR detectors. He is a physicist in the Astrophysics Department of CEA-Saclay. He is now working on infrared detectors based on HgCdTe technology for R&D programs and as the detector scientist of one of the instrument of the ARIEL space mission.

**Enrico Biancalani** is a master's student in astronomy and instrumentation at Leiden University and at TU Delft and a support astronomer at the Nordic Optical Telescope. His scientific research focuses on the technologies and on the techniques tailored for the direct imaging and spectrography of exoplanets in order to detect and characterize Earth analogues.

**James Gilbert** is a program manager at RocketLab, responsible for the delivery of avionics hardware, flight software, radio communications, and GNC for the Neutron reusable launch vehicle. His previous work at Australian National University included the development of infrared avalanche photodiode (APD) payloads for Earth observation.