



QUEENS: an open-source Python framework for solver-independent analyses of large-scale computational models – from parameter studies and identification, sensitivity analysis, surrogates, optimization, (Bayesian) forward and backward uncertainty quantification to digital twinning

Jonas Biehler^{1,3} · Jonas Nitzler¹ · Sebastian Brandstaeter² · Maximilian Dinkel¹ · Volker Gravemeier³ · Lea J. Haeusel¹ · Gil Robalo Rei¹ · Harald Willmann¹ · Barbara Wirthl¹ · Wolfgang A. Wall¹

Received: 21 November 2025 / Accepted: 21 April 2026
© The Author(s) 2026

Abstract

A growing challenge in research and industrial engineering applications is the need for repeated, systematic analysis of large-scale computational models, for example, patient-specific digital twins of diseased human organs: The analysis requires efficient implementation, data, resource management, and parallelization, possibly on distributed systems. To tackle these challenges and save many researchers from repetitive, time-consuming tasks, we present *QUEENS* (Quantification of Uncertain Effects in Engineering Systems), an open-source Python framework for composing and managing simulation analyses with arbitrary (physics-based) solvers on distributed computing infrastructures. Besides simulation management capabilities, *QUEENS* offers a comprehensive collection of efficiently implemented state-of-the-art algorithms ranging from routines for convergence studies and common optimization algorithms to more advanced sampling algorithms for uncertainty quantification and Bayesian inverse analysis. Additionally, we provide our latest cutting-edge research in multi-fidelity uncertainty quantification, efficient multi-fidelity Bayesian inverse analysis, and probabilistic machine learning. *QUEENS* adopts a Bayesian, probabilistic mindset but equally supports standard deterministic analysis without requiring prior knowledge of probability theory. The modular architecture allows rapid switching between common types of analyses and facilitates building sophisticated hierarchical algorithms. Encouraging natural incremental steps and scaling towards complexity allows researchers to consider the big picture while building towards it through smaller, manageable steps. We aim to create a large scientific community that seeks to collect, extend, and develop methods at the intersection of physics-based simulations, data-driven machine-learning techniques, and sophisticated simulation analytics. The open-source repository is available at <https://github.com/queens-py/queens>.

Keywords Open-source software · Parameter identification · Uncertainty quantification · Sensitivity analysis · Bayesian inverse analysis · Optimisation

1 Introduction

Computational models have become indispensable in advancing scientific and engineering disciplines. These models encompass many facets, such as hardware dependencies, physical model assumptions, algorithmic

complexity, numerical configurations, and various (uncertain) parameterizations.

Researchers often seek to scrutinize and optimize various aspects of computational models and their setups. This iterative process can quickly become cumbersome and often includes formidable challenges and infrastructure

Jonas Biehler, Jonas Nitzler and Sebastian Brandstaeter contributed equally to this work.

Extended author information available on the last page of the article

constraints. Moreover, many sources of uncertainty exist, including model errors, incomplete information, and inherent variability in the studied system. Understanding and quantifying these uncertainties become vital prerequisites for informed decision-making.

Consequently, a wide range of libraries for different algorithms and types of model analysis, such as PyDOE [1], SALib [2], Chaospy [3], UQLab [4], UQPy [5], CUQIPy [6, 7], ELFI [8], PyMC [9], and particles [10] have emerged. While each of these tools makes valuable contributions, our experience in developing and applying multi-query analysis methods for large-scale computational models has revealed a specific combination of challenges that remains insufficiently addressed. Algorithm-focused libraries such as Chaospy, SALib, PyMC, or UQLab provide mature implementations for method development and testing. However, transitioning a newly developed method from a prototype tested on analytical or lightweight models to production use with computationally expensive, externally executed simulations on HPC clusters typically requires substantial re-implementation. Conversely, simulation management frameworks such as Dakota, OpenTURNS, UQpy, or EasyVVUQ offer varying degrees of solver coupling and HPC integration, but their architectural constraints—whether a C++ codebase requiring recompilation for new algorithms (Dakota), limited Bayesian inference capabilities (OpenTURNS, EasyVVUQ), or commercial licensing

requirements (UQLab)—can hinder the rapid development cycle that computational researchers require. UM-Bridge contributes an elegant language-agnostic coupling protocol but deliberately provides no methods or workflow management of its own.

Our novel framework *QUEENS* (Quantification of Uncertain Effects in Engineering Systems) is an open-source Python framework designed to address this gap by combining three capabilities that, to the best of our knowledge, no single existing framework provides simultaneously: (i) a Python-native environment for algorithm development with full access to the modern scientific computing and machine learning ecosystem, (ii) robust and automated coupling with arbitrary external simulation codes on distributed HPC infrastructures, and (iii) a composable, nestable workflow architecture that supports hierarchically complex analyses—from simple parameter studies to surrogate-assisted Bayesian multi-fidelity inverse analysis—without requiring re-implementation when scaling towards greater complexity. A further challenge lies in the efficient orchestration of two-level parallelism: at the outer level, many independent model evaluations must be scheduled concurrently, while at the inner level, each individual simulation may itself require distributed execution across multiple processors or nodes. The optimal allocation of computational resources between these two levels depends on the analysis algorithm

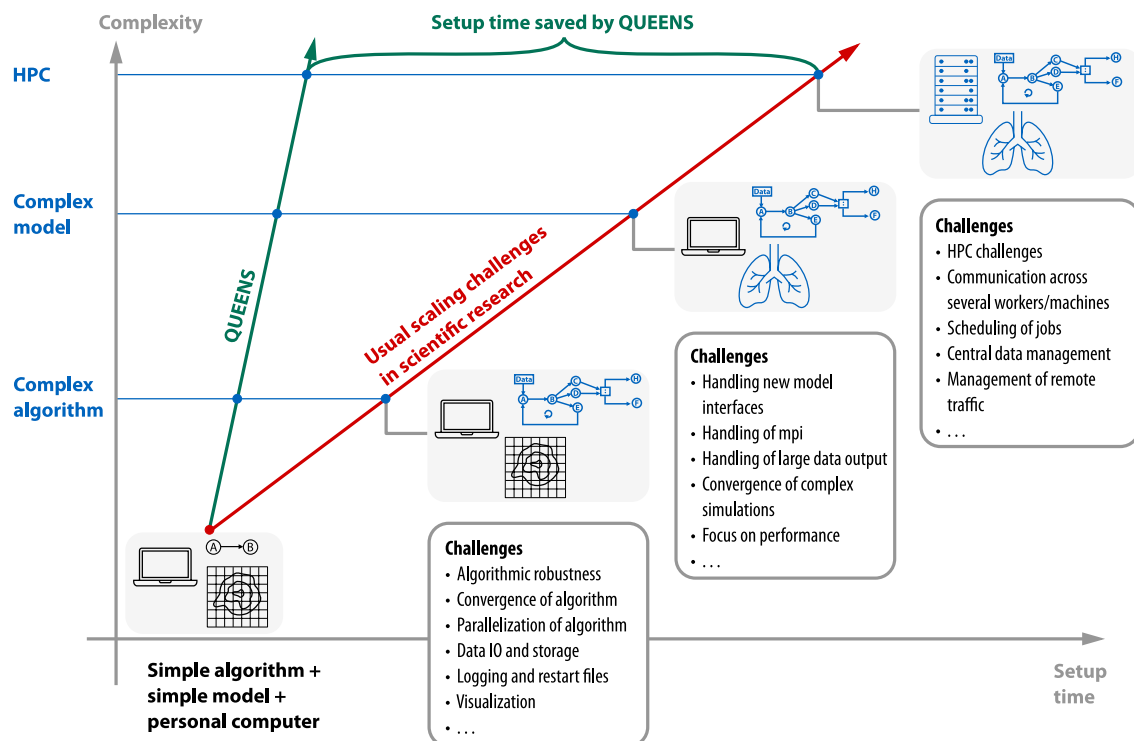


Fig. 1 Usual challenges encountered when scaling scientific research workflows from *personal computer - simple algorithm - simple simulation model* to *distributed computing - complex algorithm - large*

scale, complex simulation models. The complexity scaling is exemplified for the order: complex algorithm - complex model - HPC, but should be understood interchangeably

and the available hardware, and *QUEENS* provides the infrastructure to flexibly manage this trade-off.

First and foremost, we developed *QUEENS* to provide a framework that facilitates a fast setup for simulation analysis that can easily scale from the personal laptop to distributed HPC workflows while taking care of all simulation and data management. Secondly, *QUEENS* is a computational platform to maintain, develop, and share algorithmic advances in optimization, uncertainty quantification, sensitivity analysis, (Bayesian) inverse analysis (backward uncertainty quantification), and simulation analytics. Most users of *QUEENS* might use the provided abstraction levels to design their simulation analysis pipelines with their own or commercial third-party simulation frameworks, such as finite element software, particle methods, or any other numerical discretization approach. In this context, we use the terms finite element simulations or physics-based simulations as umbrella terms that should include all approaches for discretized PDE solvers. For this purpose, we already offer many state-of-the-art algorithms and our own developments (see Sect. 2 for an overview). Additionally, we also want to encourage active method development. A core principle of *QUEENS* is encouraging natural incremental scientific steps towards more complexity, which allows researchers to develop complex algorithms and workflows robustly. The study of arbitrary simulation codes requires robust infrastructure and interfaces, powerful packages like *QUEENS* or similar state-of-the-art frameworks, such as UM-Bridge [11], Dakota [12], OpenTURNS [13], and EasyV-VUQ [14] have proven instrumental in meeting these needs. *QUEENS*, along with these packages, capitalizes on a common fundamental characteristic shared by various model analyses: the repetitive evaluation of the system of interest, often in a batch sequential setting. In this context, the workflow alternates between sequential steps that are interdependent and parallel steps that can be executed simultaneously. Specifically, the parallel workload typically involves independent evaluations of the physical model, lending itself to an embarrassingly parallel algorithm that optimizes computational resources. We define such a setting in a broad sense as *multi-query tasks*, i.e., the efficient evaluation of model output quantities of interest, enabling the simultaneous exploration and characterization of the system under investigation.

When setting up analysis pipelines for existing simulation frameworks, such as finite element codes, researchers usually start their work on a personal workstation or laptop using a simplified version of the simulation model and a first draft of an algorithmic implementation for the model study. Implementing this first simple workflow is usually straightforward, but specialized to the problem at hand, needing more flexibility and abstraction for building up to more challenging workflows. Nevertheless, the goal is primarily to scale an existing setup to an HPC system using the full high-fidelity physical

simulation model. Elevating the research setup to the subsequent complexity level will require many manual steps, such as updating scripts, changing interfaces, and restructuring code and algorithms to facilitate the more complex requirements. Figure 1 represents some examples of time-consuming pitfalls and challenges when setting up typical research stages. Schematically, we depict the workflow complexity over the setup time required to reach a robust implementation at the desired complexity level. The significant refactoring and restructuring are prone to errors and, therefore, require extensive testing and validation at every scaling stage. By scaling, we refer to increasing the complexity of the three main components: the transition from small to extensive simulations, from simple to complex algorithms, and from laptops to HPC distributed computing systems.

QUEENS accelerates this process as it already implements mature abstraction layers to allow for scaling to higher levels of complexity. Researchers can, therefore, focus on algorithmic implementations and computational simulations and need to worry less about simulation handling, data management, and interfaces.

We summarize our intentions behind *QUEENS* in the following *mission statement*:

Mission statement

Our mission is to provide a comprehensive computational platform for the design and execution of multifaceted simulation analysis tasks involving data-driven to physics-based computational models. We aim for a high-level abstraction to facilitate rapid prototyping. The platform supports diverse methods, from convergence studies to advanced probabilistic analysis. QUEENS enables incremental algorithm development based on basic Python models to workflows involving sophisticated large-scale (e.g., finite element) simulations on HPC platforms. Robust implementations and automated simulation execution should free researchers to focus on their research objectives. We strive for sustainability through the open-source community.

The remainder of the paper is structured as follows: We first clarify the target audience and application domains for *QUEENS* and provide a short overview of already existing algorithms in Sect. 2. Afterwards, we outline the architecture and design principles behind the *QUEENS* software framework in Sect. 3. We also show how the modular design facilitates the sequential build-up of complex analysis pipelines. Eventually, in the hands-on Sect. 4, we demonstrate with the help of two already published examples how a *QUEENS* simulation can be constructed from scratch. The examples cover a parameter study for a biomechanical tumor growth

model and the Bayesian calibration of a fluid–structure interaction problem. Section 5 provides a short conclusion and outlook for the software project *QUEENS*.

2 Overview of currently available methods and target users

Before delving into more detailed architectural aspects, we briefly delineate application domains and the target audience for *QUEENS*. The software is designed for both applied users as well as computational researchers who require an automated framework for analyzing existing simulation frameworks. Typical applications range from simple convergence or automated parameter grid studies to complex optimization tasks or nested probabilistic models and Bayesian inverse analysis in combination with several finite element codes and surrogate models. *QUEENS* is ideal for users who want to build modular analysis tasks for their existing simulation frameworks (for example, a physics-based finite element model) while having the possibility to

scale their framework quickly from laptop to HPC clusters. The software simplifies the integration of analysis methods with existing legacy codes and manages resources without the user needing to establish these systems from scratch. A distinctive strength of *QUEENS* is its integrated support for advanced statistical and probabilistic analysis. We enable users to quickly construct probabilistic models and probabilistic machine learning tasks in combination with large numerical simulation frameworks, run uncertainty quantification (UQ) and Bayesian inverse analysis, and integrate surrogate models. In this context, *QUEENS* is also a suitable framework for *digital twins* in engineering, industry, and science. By *digital twin*, we mean a virtual representation of a (physical) system (i.e., a physics-based simulation model discretized by the finite element method) with a bidirectional interaction (i.e., repeated flow of data) between virtual representation and real world). In such a context, a Bayesian calibration process conducted with *QUEENS* can then, for example, update uncertain or unknown parameterizations of the simulation model to reflect the real-world process while accounting for the present uncertainties. The calibrated model can be used for predictive purposes in a subsequent step and can lead to changes in the real system. *QUEENS* focuses on efficient parallelization of all mathematical operations and data management.

We currently already offer many state-of-the-art algorithms, ranging from surrogate modeling techniques to UQ to optimization and inference algorithms. We also provide our developments in probabilistic machine learning, UQ, and Bayesian inverse analysis for large-scale simulations. Table 1 summarizes a collection of currently available methods. As *QUEENS* is an actively developed code, an up-to-date state of the software can be found in our repository at <https://github.com/queens-py/queens> under the GNU Lesser General Public License (LGPL).

The methods and algorithms are designed with well-defined interfaces, which makes it very easy to change, combine, or nest different analysis types to build up more complex and flexible algorithms. We will go into more detail on the software architecture's modularity, scalability, and simplicity in the subsequent Sect. 3.

First steps and simpler types of analyses (Convergence and grid studies, surrogates) A starting point for new *QUEENS* users might be a small convergence study that evaluates, e.g., a finite element model for different FE mesh sizes or polynomial degrees, and afterwards plots an error estimate of the quantity of interest. Other simpler workflows include a two- or three-dimensional parameter study over a parameter grid or specified design of experiments, such as a Latin hypercube design. Section 4 gives a hands-on example of such an analysis. *QUEENS* evaluates the underlying FE model automatically and in parallel

Table 1 Overview of available methods in *QUEENS*

Algorithm/method	References
Design of experiments	
Grid designs	
Convergence studies	
Monte Carlo sampling	[15]
Latin hypercube sampling	[16]
Sobol sequence	[17]
Optimization	
Levenberg-Marquardt algorithm	[18, 19]
SciPy optimizers (Conjugate gradient, BFGS,...)	[20]
Stochastic optimizers (Adam, Adamax, RMSProp)	[21, 22]
Uncertainty Quantification	
Monte Carlo sampling	[15]
Bayesian multi-fidelity Monte Carlo (BMFMC)	[23–26]
Polynomial chaos expansion (PCE)	[3]
Sensitivity analysis	
Elementary effects method	[27]
Sobol method	[28, 29]
Sobol method with surrogate uncertainties	[30, 31]
Surrogates	
Gaussian processes (variational, heteroskedastic, etc.)	[32, 33]
(Bayesian) Neural networks	[34]
Gaussian neural networks / inference networks	
Polynomial chaos surrogates	[3]
(Bayesian) Inverse Analysis (BIA)	
Markov Chain Monte Carlo	[15, 35]
Sequential Monte Carlo	[10, 36]
Stochastic variational inference (reparameterization trick)	[37–41]
Black box variational inference	[37, 42, 43]
Bayesian multi-fidelity inverse analysis	[44]
Adaptive and constrained GPs for BIA	[45]

for all combinations of given mesh sizes and polynomial degrees. Subsequently, the user can directly train and visualize a surrogate model (e.g., a Gaussian Process model), which can again be used for further analysis.

Intermediate types of analyses (UQ, sampling methods, optimization, gradient calculations) More elaborate analyses, such as optimization, uncertainty quantification (UQ), or global sensitivity analysis, can also be conducted. For optimization problems, *QUEENS* integrates the optimization algorithms provided by SciPy [20], such as the conjugate gradient method, BFGS, and the Levenberg–Marquardt algorithm [18, 19], while taking additional care of embarrassingly parallel tasks and the computation of model gradients. Especially for the latter, we offer a sophisticated abstraction level that enables different ways of computing model gradients via finite difference schemes, automated differentiation (if supported by the simulation model), or incorporating adjoint models to exploit adjoint-based gradient calculation. Additionally, *QUEENS* supports stochastic optimization techniques, such as Adam [21], Adamax [21], and root mean square propagation (RMSProp) [22]. For UQ, *QUEENS* provides different approaches to quantify the uncertainty of model outputs: besides simple but potentially expensive Monte Carlo methods [15], *QUEENS* also supports polynomial chaos expansion (PCE) [3] to reduce the number of model evaluations. If a lower fidelity model is available, e.g., a model with fewer degrees of freedom or simplified physics, we also provide our own efficient Bayesian multi-fidelity Monte Carlo method (BMFMC) [23–26] to reduce computational cost drastically. Another intermediate analysis type is the global sensitivity analysis. *QUEENS* includes global approaches, namely the Elementary Effects method (also called Morris method [27]) and the Sobol method [28, 29], to apportion the uncertainty in the model output to different sources of uncertainty in the model input [46]. Due to the modular design, most methods in *QUEENS* can use surrogate models to approximate the investigated model, reducing the number of potentially expensive model calls. Available regression models range from heteroskedastic Gaussian processes to Bayesian neural networks. The modularity of *QUEENS* makes it easy to change the analysis type while investigating physics-based simulations, such as finite element models of engineering problems.

More complex types of analyses (Bayesian inverse analysis and nested probabilistic models) More complex algorithmic setups include methods for Bayesian inverse problems in combination with distributed, large-scale physics-based (finite element) models. *QUEENS* provides a wide range of state-of-the-art inference methods such as Markov Chain Monte Carlo [15] and Sequential Monte Carlo [10, 36] or gradient-based stochastic variational inference

[37–41] for which the *QUEENS* internal gradient abstraction can be used to calculate the model gradients.

Own research in forward and backward uncertainty propagation In addition to state-of-the-art methods, *QUEENS* allows us to share our research in Bayesian inverse analysis, uncertainty quantification, optimization, probabilistic machine learning, and global sensitivity analysis tailored to large-scale simulation models. This approach enables other researchers to quickly implement and build upon our methods, fostering collaboration and accelerating progress in the field. The interested reader is encouraged to look into the following publications that present a more in-depth use of *QUEENS* for complex applications outside this paper’s scope. We, e.g., provide novel and very efficient variants of black-box variational inference algorithms [37, 42, 43] that do not require model gradients or novel adaptively learned probabilistic surrogates [45] to approximate the posterior distribution directly and very efficiently. For efficient uncertainty quantification, we developed Bayesian multi-fidelity Monte Carlo algorithms [23] based on the authors’ previous work [24–26]. *QUEENS* also offers methods for global sensitivity analysis in combination with challenging, large-scale biomechanical models [31, 47]. Some other applications of *QUEENS* include calibration problems involving microstructure development in selective laser melting [48, 49], calibration of constitutive laws for biofilms [50], and tumor-model calibration [51], using the Levenberg–Marquardt algorithm. Please also see our publication on Bayesian calibration under uncertainty, when only deformed interface information is accessible [52], or the Bayesian calibration of an advanced tumor model with actual experimental data [53].

For an up-to-date overview of our publications with *QUEENS* and further news, please visit our website <https://www.queens-py.org>.

3 Software architecture and design principles

To allow for a convenient setup of the algorithms above in combination with a wide range of (physics-based) simulation codes on different computing infrastructures, *QUEENS* follows four design principles: *Modularity*, *Scalability*, *Extensibility*, and *Simplicity*. In the following, we give a summary of these principal ideas:

Modularity (flexible and straightforward build-up of workflows from simple blocks) We identified an appropriate module abstraction level that balances minimal implementation effort and maximum flexibility. The latter allows for easy substitution of components, thereby enhancing the adaptability of the overall system. Moreover, it enables the

creation of more elaborate nested analysis workflows and encourages using various solvers, software, hardware, and methods to meet specific requirements. On the highest abstraction level, we distinguish the *Algorithm Layer*, the *Model Layer*, and the *Execution Layer*, which represents a collection of several Python modules in the software architecture. Figure 2 overviews these principal design layers, including their sub-components and dependencies.

As depicted in Fig. 2, the architecture follows a top-down delegation pattern with well-defined interfaces between layers. At the top, the *Parameter* module encapsulates uncertain inputs as probability distributions, exposing sampling

routines, density evaluations, and gradient computations to the layers below.

The *Algorithm Layer* centers on the *Iterator* abstraction, which orchestrates the outer loop of a given method, such as Monte Carlo sampling, optimization, or variational inference, by requesting parameter samples, dispatching model evaluations, and aggregating results. Iterators are composable: a surrogate-assisted Bayesian optimization workflow, for instance, nests a training iterator inside the surrogate model, which itself is queried by an outer optimization iterator.

The *Model Layer* exposes a uniform `evaluate` interface through three complementary abstractions. A *Simulation*

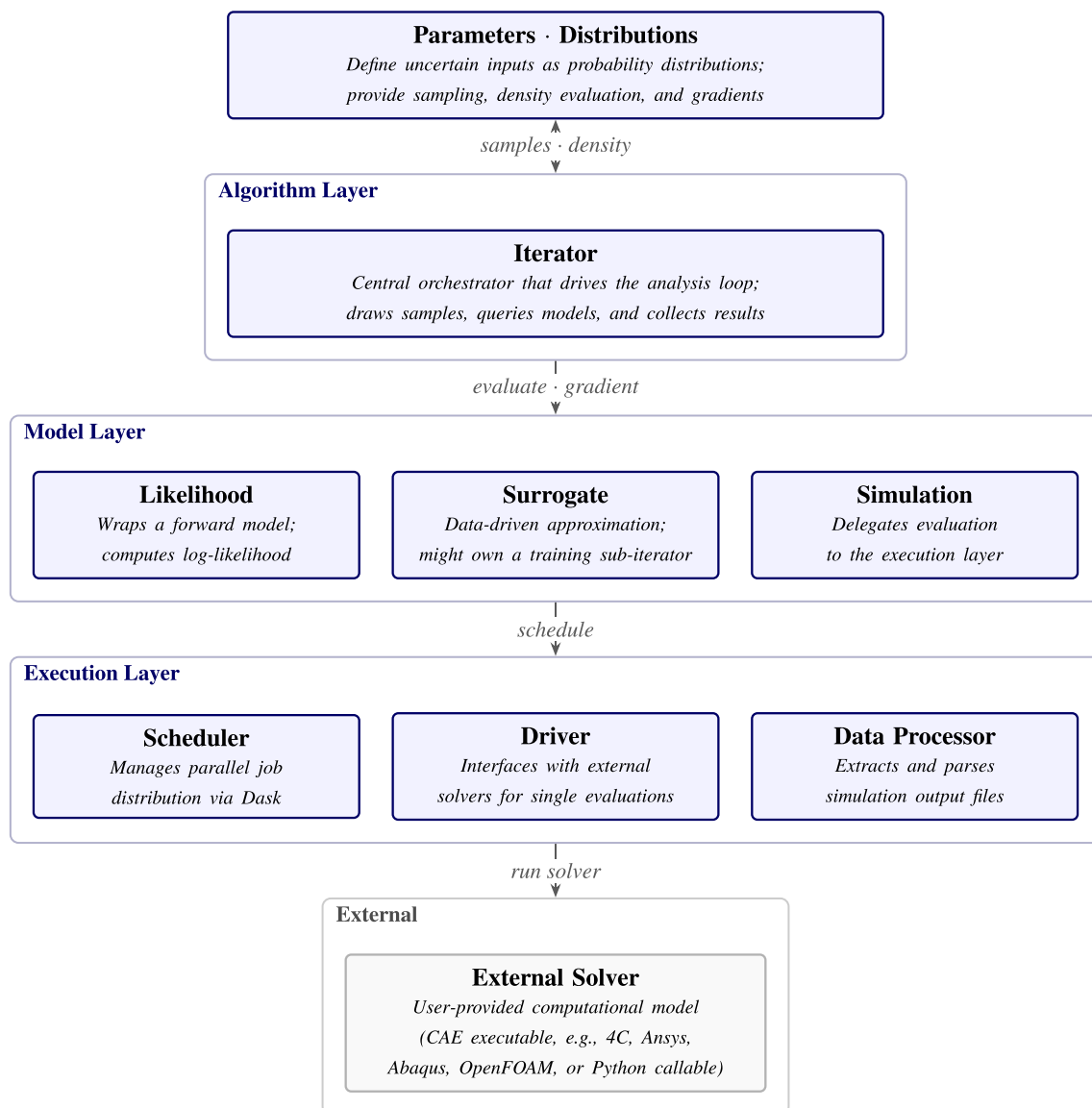


Fig. 2 Overview of the *QUEENS* software architecture. The *Algorithm Layer* contains iterators that orchestrate analysis workflows. The *Model Layer* provides uniform evaluation interfaces through likelihood models, surrogates, and simulation wrappers. The *Execution Layer* man-

ages scheduling, driver communication with external codes, and data processing. Parameters and distributions define the uncertain inputs at the top; external solvers provide the computational backend at the bottom. Arrows indicate primary data flow and delegation between layers

model wraps a forward solver and delegates its execution to the layer below. A *Likelihood* model composes a forward model with an observation model to compute log-likelihood values and, optionally, their gradients. A *Surrogate* model provides a data-driven approximation (e.g., a Gaussian process) that conforms to the same interface, rendering it interchangeable with a full-fidelity simulation. This substitutability is enforced through a common abstract base class.

The *Execution Layer* decomposes each model evaluation into input-file generation, job scheduling, solver invocation, and output parsing. The *Scheduler* manages parallel job distribution through a Dask [54] backend, supporting both local execution and remote HPC submission (e.g., via SLURM). The *Driver* handles communication with external solver processes, and the *Data Processor* extracts quantities of interest from raw output files (e.g., VTK, CSV, or HDF5) and maps them into the internal data representation. This decomposition ensures that adding support for a new solver typically requires only a new driver and data processor.

From the user perspective, most implementations and analyses are carried out at the Algorithm and Model Layers using *iterators* and *models*, which are the building blocks for creating complex algorithms involving multiple methods and models. The Execution Layer handles computation management transparently, as described above, allowing users to focus on algorithmic design rather than solver integration.

The Execution Layer is also responsible for archiving, storing, and making simulation data available through its data processing components. These manage data formats and internal variable structures, handle I/O tasks, ensure data is available across multiple platforms and computing resources, and handle metadata logging.

Figure 3 shows how a Monte Carlo analysis of a Python function can be easily changed into a complex Bayesian inverse analysis of an actual large-scale external finite element model by extending and switching building blocks of the analysis. For clarity, we briefly comment on some of the *QUEENS* internal terms: *Method* denotes the (iterative) algorithm acting on and evaluating an abstract model. The latter can be a *simulation model* (e.g., a finite element model) or can be a probabilistic model (e.g., a likelihood model), which has another finite element model nested. In principle, a *model* refers to an abstract mapping in a mathematical sense. Another term in the input file is an *interface*. This module specifies how *QUEENS* can communicate with a *model*: In principle, we distinguish Python internal models, which can be directly accessed and evaluated within the Python programming language, or external models in the form of third-party executables and codes (such as Ansys, other C++ executables, containers, etc.). Finally, the term *driver* means, in this context, a finer-grained version of the

interface module that contains settings for the type of MPI communication (local or remote) and handles the parsing of parameters to the respective input files of external executables and codes.

Analysts can quickly and independently change or reuse input blocks from previous *QUEENS* runs. All building blocks are organized logically and systematically.

Scalability (from simple algorithms on the personal laptop to complex workflows on the HPC cluster) We developed *QUEENS* to manage extensive workloads and intricate algorithms. A critical feature enabling this capability is *scalability*. In *QUEENS*, scalability appears in multiple forms, such as computational scaling from a laptop to a high-performance computing (HPC) system and algorithmic scaling from a basic parameter study to a comprehensive probabilistic analysis. Our modular design (see previous paragraph) naturally contributes to scalability. The following paragraph emphasizes key concepts and practical aspects of scalability in *QUEENS*. We intend to nurture a natural incremental workflow in sciences, where problems can be tackled step-wise, from small to large scales, and from simple to complex problems. The incremental approach is essential in handling high complexity and growing scientific and technological challenges. We utilize the well-established Python library *Dask* [54] for scheduling several jobs on a local machine or a remote HPC cluster with support for typical queuing systems such as PBS [55] or Slurm [56]. We also handle the fully distributed parallel execution of individual simulations over multiple processor cores or nodes using a message-passing interface (MPI). Scaling the computation from a single machine to an HPC cluster is easy because *QUEENS* automatically handles the tedious steps in the background: remote communication to the HPC cluster, creating the individual input files and job scripts, submitting jobs to the job queuing system, and gathering the results. We implemented customized versions of many algorithms to ensure embarrassingly parallel execution where possible. Additionally, we included extensive logging capabilities to track the progress of the analysis and simplify debugging. We achieve the scalability of the algorithmic complexity in *QUEENS* by utilizing the building blocks of the Algorithm and Model Layers to switch and especially to nest methods and models. These configurations can be easily updated in the input file, allowing for a seamless transition from simple to more complex computations with minimal modifications to the input file. An example of scaling the algorithmic complexity of uncertainty quantification (UQ) by nesting algorithmic building blocks is presented in Fig. 4.

Many methods and model concepts in *QUEENS* leverage parallelism through embarrassingly parallel algorithms and through fully distributed simulation frameworks, for example, based on the Message Passing Interface (MPI).

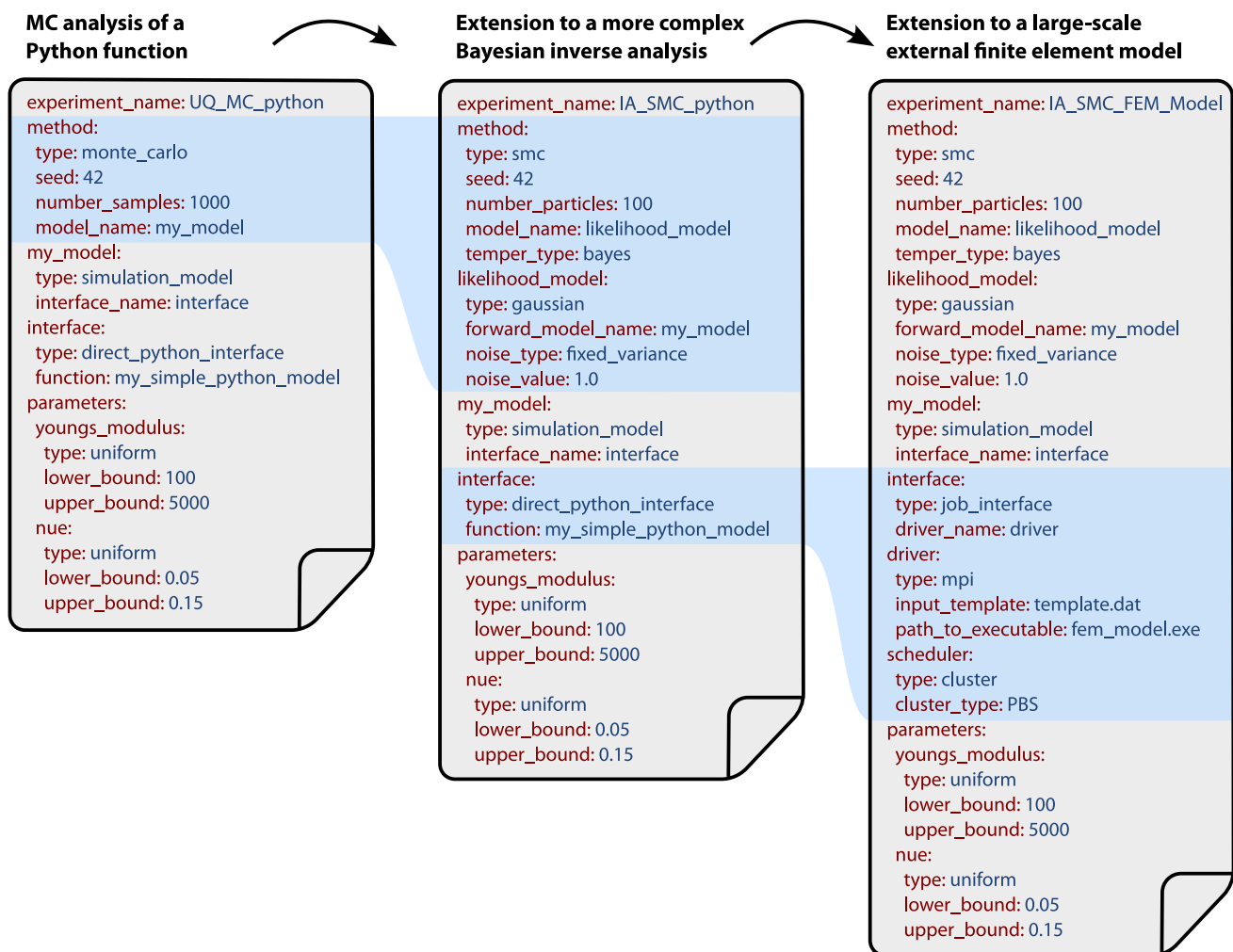


Fig. 3 Example of the modular character of a *QUEENS* input file, which allows for easy extension towards more complex algorithmic setups and types of analysis. We start from a simpler Monte Carlo (MC) analysis of a function written in Python to assess the uncertainty propagation associated with the input parameters *youngs_modulus* and *nue*. Afterwards, we extend the method or analysis type to a more

complex Bayesian inverse analysis. The blue parts of the input file mark the sections that need to be changed. Finally, we replace the Python function with an actual finite element model in the last input file. Please note that we can use any available solver by providing the appropriate interface and the path to the executable

Embarrassingly parallel workloads consist of numerous independent tasks that require no inter-task communication; scalability is achieved simply by increasing the number of concurrently executed jobs. In contrast, large-scale computational models typically rely on intra-task parallelism, where a single simulation is distributed across multiple CPUs or nodes, commonly implemented using MPI.

The workloads addressed by *QUEENS* combine both dimensions of parallelism. At the top level, analyses are task-parallel, comprising many independent jobs. Within each job, however, data- and communication-parallel execution enables scalable simulation of complex models. In practice, this spectrum ranges from hundreds of small independent tasks (we have demonstrated more than 400 concurrent jobs) to scenarios involving multiple multi-node

MPI simulations running simultaneously—for example, 16 independent jobs, each executed with 48 MPI ranks.

QUEENS is designed to flexibly support this entire range. While it naturally accommodates large ensembles of lightweight tasks as well as fully distributed large-scale simulations, it particularly excels in the intermediate mixed regime, where ensembles of moderately sized MPI jobs must be orchestrated efficiently and robustly. An illustration of these different scenarios, including real data points from scientific studies conducted with *QUEENS*, is provided in a parallelism map in Fig. 5. The diagonal lines in Fig. 5 denote constant total core counts, i.e., concurrent jobs $\times$ cores per job = const, and can be interpreted as hardware limits imposed by the available HPC resources. For a fixed allocation, feasible configurations are

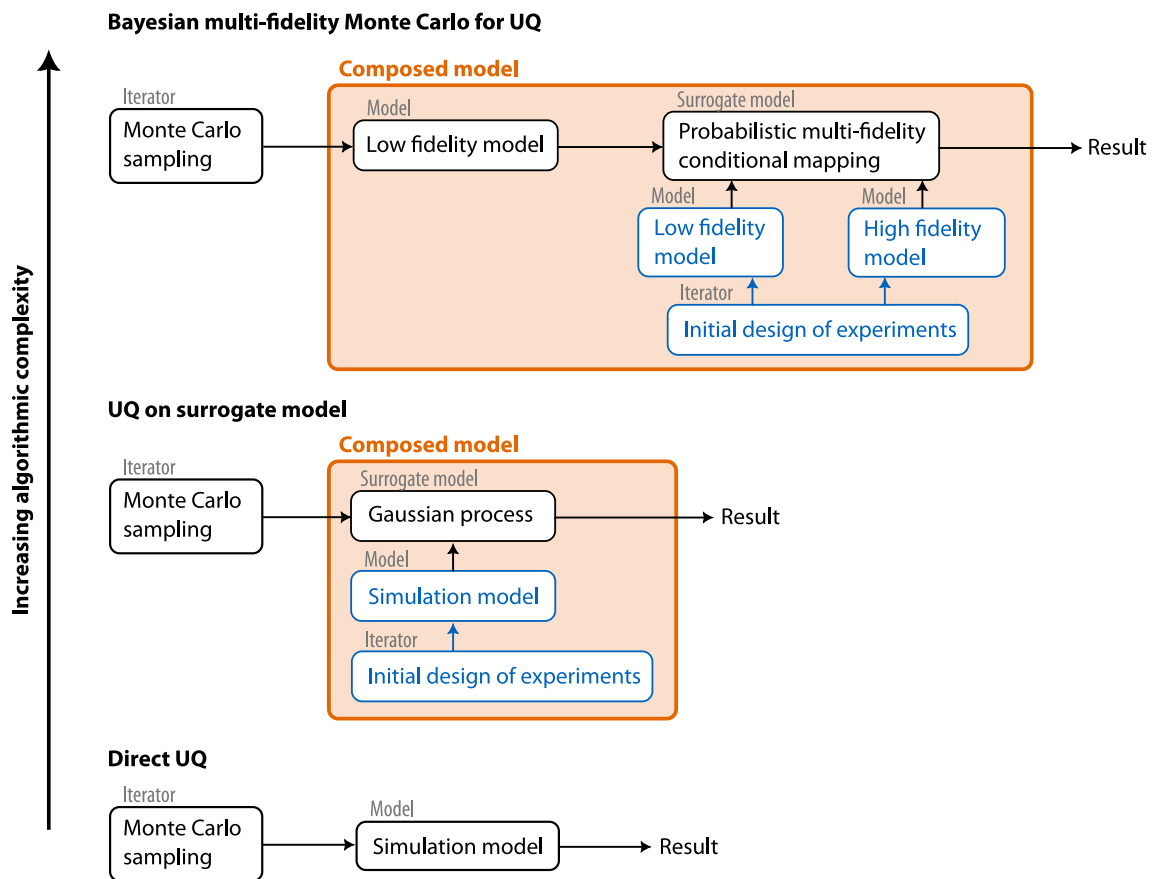
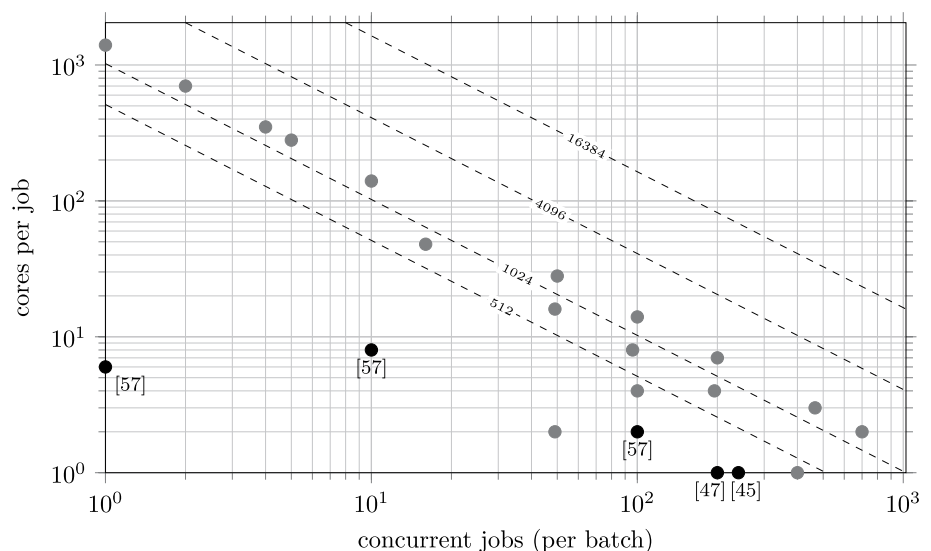


Fig. 4 Construction of nested workflows in *QUEENS* using the example of uncertainty quantification (UQ). The parts marked in blue can be performed offline before starting the UQ. In the first case (bottom), we perform direct Monte Carlo sampling on the simulation model (e.g., a finite element model) and yield the corresponding output distribution in sample representation. In the second case (middle), we replace the simulation model with a surrogate (e.g., a Gaussian process), on which we can then perform the sampling. We initialize and train the surrogate

model using an initial design iterator on the more expensive simulation model (e.g., a finite element model). Finally, in the top row, we demonstrate the setup of Bayesian multi-fidelity Monte Carlo for UQ: We first conduct Monte Carlo sampling on a low-fidelity model and then correct its response statistically by a probabilistic multi-fidelity mapping. The latter is trained on corresponding tuples of high- and low-fidelity model evaluations drawn from an initial design

Fig. 5 Parallelism map illustrating the two-level parallel structure supported by *QUEENS*: outer task-level parallelism (concurrent independent jobs) and inner solver-level parallelism (MPI ranks per job). The spectrum ranges from large ensembles of lightweight tasks to multiple multi-node MPI simulations running simultaneously. Black markers denote configurations from published studies (see cited references), while gray markers represent previously unpublished simulation setups. Diagonal lines indicate constant total core counts and thus potential hardware limits of the compute resource. Efficient execution requires balancing algorithmic task parallelism and solver scalability within these constraints



bounded by such a limit. The central challenge is therefore how to distribute resources optimally under two constraints: (i) algorithmic limits on embarrassingly parallel tasks, and (ii) solver scalability, which determines how efficiently a single forward model can exploit additional MPI ranks. If a desired configuration exceeds the hardware limit, serialization becomes necessary—either at the outer level (fewer concurrent jobs executed in batches) or at the inner level (fewer MPI ranks per job with longer runtimes). The optimal strategy depends on the analysis at hand: Monte Carlo methods typically benefit from maximizing outer-level concurrency, whereas algorithms with limited task parallelism—such as gradient-based optimization methods (e.g., CG or BFGS)—tend to rely more heavily on solver-level scalability. *QUEENS* is designed to enable users to flexibly balance these two levels of parallelism within the constraints imposed by the chosen analysis type and the available HPC system.

Extensibility (new methods, collaborations, shared research, and open-source documented scientific code) We want the software platform to accommodate future changes and additions, such as adding new functions or modules, without requiring extensive modification of its existing code base. In the context of Python, the extensive libraries and modules available in its ecosystem and its simplicity and readability make it easy for developers to add new submodules and functions, making the software more extensible. Extensibility for data-driven, probabilistic machine learning and optimization algorithms with many sophisticated and specialized packages is available and under continuous development by the Python community and can be easily included in *QUEENS*.

Furthermore, the *QUEENS* developers are committed to continuously extending and enhancing the software. We see *QUEENS* as a scientific hub where our research is shared, and we encourage other researchers to contribute their methods. In this context, *QUEENS* provides a collaborative scientific platform where researchers can make their methods openly available in the public repository at <https://github.com/queens-py/queens>. This helps researchers comply with data sharing policies, for example, when publishing papers, accelerates the pace of scientific discovery, and facilitates greater transparency and reproducibility in research.

Simplicity (non-expert user to professional method developers) We designed *QUEENS* to be easy to use and to promote understanding through clean syntax, precise and extensive documentation, modular design with well-defined interfaces, and availability of pre-built modules and libraries. These abstraction levels make it easy to set up complex workflows and allow straightforward extension and customization with new methods and modules, fostering accessibility and maintainability.

4 Hands-on examples

We now give some practical insights on using *QUEENS*: We present the setup of a deterministic parameter study of a tumor-growth model [53] and a Bayesian calibration of a biofilm model [52]. The algorithmic building blocks can be defined in the *QUEENS* input file in arbitrary order and then extended as needed. The internal setup takes care of the dependencies within the algorithm.

4.1 Parameter study of a tumor-growth model with *QUEENS*

In this example, our goal is a parameter study of a tumor-growth model. Figure 6 visualizes the *QUEENS* setup for this example.

We start by setting up our forward problem. In this case, we have a tumor growth modeled as a multiphase porous medium and solved with the finite element method (FEM) as, for example, presented in [53]. We want to investigate the tumor volume after seven days of growth, depending on a variation of two input parameters: the growth coefficient γ_{growth} and the surface tension σ_{tl} . In the *QUEENS* input file, we define the tumor-growth FEM model in the **forward model block** as shown in Fig. 6A. This block configures the type of model (here, a simulation model), the associated parameters (which we will subsequently define in a separate block), and the interface to the simulation software. The latter holds paths to the executables and input files for the FEM simulation and other settings, such as MPI and parallelization configurations.

In the **parameter block** (detailed in Fig. 6B), we configure the input parameters. We define the ranges that we want to study for the two parameters. *QUEENS* uses the parameter names from the input file and searches for placeholders with the same name in the template input file of the FEM simulation. It then parses the actual values for these parameters during the *QUEENS* run to start a new FEM simulation for each parameter combination.

Finally, the **method block** (detailed in Fig. 6C) defines which method with which settings to use. For our parameter study, we use the grid iterator and specify that we want to evaluate a 10×10 grid for the two input parameters.

The input file has two more building blocks (which are not shown in Fig. 6): A block for global settings, where we set the high-level configurations of the *QUEENS* run, such as directory paths and temporary directories, and a block for the data management and job scheduling in analogy to Fig. 3.

After the setup, we start the *QUEENS* run and let the software configure and build up the workflow. A live update of which analysis section is currently running is displayed on

Goal: Parameter study of a tumor-growth model**A) Forward model block**

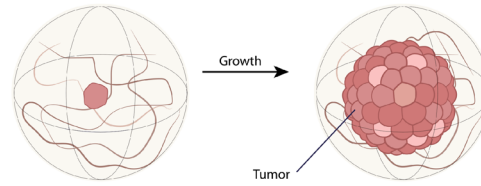
```
...
tumor_model:
  type: simulation_model
  interface_name: interface
interface:
  communication to the
  forward problem
parameters:
  description of the parameters
  gamma_growth and
  sigma_tl
...
```

B) Parameter block

```
...
parameters:
  gamma_growth:
    lower_bound: 1e-9
    upper_bound: 2e-8
  sigma_tl:
    lower_bound: 10
    upper_bound: 50
...
```

Forward problem:

Tumor-growth model as a multiphase porous medium

**C) Method block**

```
...
method:
  type: grid_iterator
  model: tumor_model
  grid_design:
    gamma_growth:
      number_grid_points: 10
    grid_type: linear
    sigma_tl:
      number_grid_points: 10
    grid_type: linear
  further settings
...
```

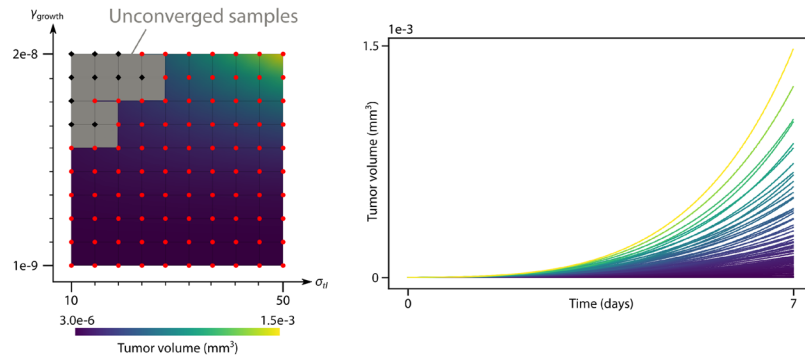
D) Output: tumor volumes for all parameter combinations

Fig. 6 *QUEENS* setup for a parameter study of a tumor-growth problem. The red dots in subfigure D) represent converged simulations

the terminal output. The analysis results are saved as a pickle file containing the evaluated input parameter combinations with the resulting tumor volumes. If desired, *QUEENS* can plot and visualize parts of the analysis depending on the settings and flags set in the input file.

In our example, the output (Fig. 6D) shows that the largest tumor volume is associated with the largest value for both the growth coefficient and the surface tension. We additionally find that for several parameter combinations, the FEM simulation did not converge: Convergence is not reached for small values of the surface tension combined with large values of the growth coefficient, marked in gray in Fig. 6. Hence, our parameter study directly included parts of a convergence study. We can also visualize the tumor volume over time for all parameter combinations to gain more insight into the results of our analysis.

4.2 Bayesian calibration of a fluid–structure interaction problem with *QUEENS*

We now discuss the setup of a Bayesian calibration problem for a coupled biofilm finite element simulation based on a recent publication of the authors [52]. Figure 7 presents an overview of the input file setup and the workflow.

We can formulate the Bayesian calibration problem as follows: Given is a pipe or channel through which a fluid

flows with an (uncertain) velocity profile. A biofilm grows on the lower boundary of the channel. A camera installation lets us observe the biofilm's boundary or interface deformation due to the surrounding fluid flow. This kind of scenario has applications in many scientific and industrial settings. For further analysis, we are interested in the constitutive properties of the biofilm, e.g., to make predictions when pipes in chemical industry plants clog due to biofilm growth. We have a coupled fluid-solid interaction finite element problem that virtually describes the scenario. The solid domain has a constitutive model with unknown Young's modulus E and Poisson's ratio ν . We want to infer these parameters based on the experimentally measured interface deformations. Figure 7 overviews the problem and the corresponding Bayesian inverse problem setup in *QUEENS*. The figure guides the analysis setup by input file snippets that contain pseudo code and explanatory plots corresponding to these building blocks. It is easiest to start from the computational forward model, which is what we do in this example: As shown in Fig. 7A, we define the biofilm model in the **forward model block**, similar to the previously discussed tumor-growth model example. The **parameter block** defines the uncertain input parameters (Young's modulus E and Poisson's ratio ν) and their prior distributions (see Fig. 7B). We here use uniform prior distributions.

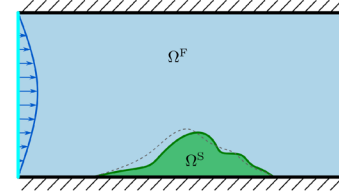
Goal: Bayesian calibration of a biofilm constitutive model**A) Forward model block**

```
...
biofilm_model:
  type: simulation_model
  interface_name: interface
  interface:
    communication to the
    forward problem
  parameters:
    description of the random
    variables  $E$  and  $v$ 
...
```

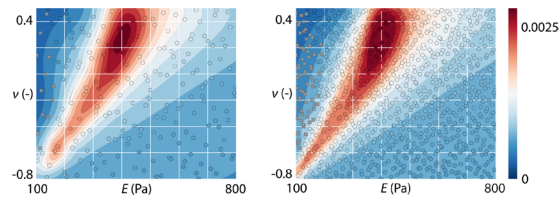
B) Parameter block

```
...
parameters:
  youngs_modulus:
    type: uniform
    lower_bound: 100
    upper_bound: 5000
  nue:
    type: uniform
    lower_bound: 0.05
    upper_bound: 0.15
...
```

Forward problem:
Biofilm as fluid-solid interaction

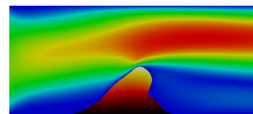
**C) Likelihood model block**

```
...
likelihood_model:
  forward_model:
    biofilm_model
  probabilistic settings:
    noise, density
    assumption,
    observation data
...
```

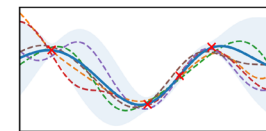
Likelihood model**D) Surrogate model block**

```
...
surrogate_model:
  training data generation
  with the likelihood_model
  and a Sobol sequence
  surrogate model attribute
  setup
...
```

Generate space-filling training
data for the Gaussian process
(e.g., with a Sobol sequence)



Introduce a Gaussian process as
surrogate model for the likelihood

**E) Method block**

```
...
method:
  type: smc
  model: surrogate_model
  prior: parameters
  further SMC settings
...
```

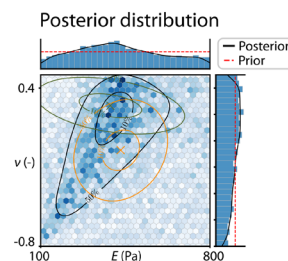
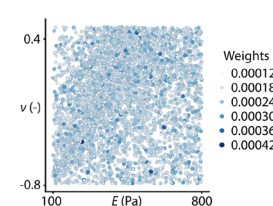
F) Output:**Particle and weight distribution from SMC**

Fig. 7 *QUEENS* setup for a Bayesian calibration of a coupled fluid–structure interaction problem (FEM simulation) using a Gaussian process meta-model to mitigate the computational costs. Figures on the

right (except the Gaussian process in the middle) are adapted from [52], licensed under a Creative Commons Attribution 4.0 International License (<https://creativecommons.org/licenses/by/4.0/>)

In the next step, we define the likelihood model for the inverse analysis in the **likelihood model block**, as detailed in Fig. 7C. *QUEENS* offers several types of likelihood models. In this example, we choose a standard Gaussian likelihood with fixed and static noise variance. *QUEENS* offers more options, such as adaptive likelihood noise with iterative averaging, which we omit here for simplicity. The likelihood model block furthermore requires a path to an

external file (e.g., a CSV file) that holds the experimental measurements along with their coordinates as observation data. We also specify which forward model the likelihood model is based upon. In our case, it is the previously defined biofilm FEM model. Together with the parameter block, which provides the prior model, this defines the full probabilistic model for the Bayesian inverse analysis, according to Bayes' rule.

As evaluating the likelihood model triggers an expensive forward simulation of the coupled FEM biofilm model, we choose to replace the mapping of the likelihood model with a cheaper-to-evaluate surrogate model. *QUEENS* offers a wide range of possible surrogate models. Here, we select a Gaussian process model, which we set up in a separate **surrogate model block** (see Fig. 7D). Within this block, we provide an iterator that specifies how (with which method and from which model) and how many training data points shall be generated to train it. We can, e.g., select a space-filling iterator, such as the Sobol sequence, set the number of training points to 500, and specify the likelihood model as the target for the training points. Once we have started the *QUEENS* run, the software allocates the resources we specified for the FEM model (the building block of the biofilm model) and runs simulations in parallel until it has realized the initial training data. Afterwards, the surrogate is trained according to other settings, such as the selected optimizer, chosen kernel model, and mean function. During the analysis, *QUEENS* can also provide metrics that assess the surrogate's quality to refine the latter if necessary. Note that in *QUEENS*, simulations models, surrogate models, and likelihood models are all models a method can act on.

Eventually, we can finish the algorithmic setup by defining the **method block** as shown in Fig. 7E. Here, we choose a sequential Monte Carlo method (SMC) to solve the Bayesian inverse problem. We set the Gaussian process meta-model as the model the iterator should act upon, and also hand over the parameter block for the prior definition. We also configure method-specific settings, such as tempering strategy, effective sample size (ESS), number of particles, and MCMC chains.

After the setup, we can start the *QUEENS* run and let the software configure, build up, and run the complex workflow. The final result, in this case, the posterior distribution of the uncertain Young's modulus E and Poisson's ratio ν will be saved as a pickle file in a sample representation along with valuable statistics and further meta-data and information, partly depending on the individual module setups in the input file. Figure 7F presents a possible output visualization.

5 Conclusion

We presented *QUEENS* as a novel comprehensive software platform for coordinating and composing complex analysis workflows for arbitrary simulation software. *QUEENS* is an open-source Python framework built on modularity, scalability, extensibility, and simplicity. Together with the variety of deterministic and probabilistic methods in combination with arbitrary forward models, this encourages the use and further development of *QUEENS* in the research

community and industry. Thus, researchers circumvent time-consuming pitfalls and challenges when setting up a complex workflow, such as running a complex probabilistic algorithm with a computationally expensive forward model on an HPC cluster.

In the coming years, we will continually update the software with a focus on our research in probabilistic analysis. We will make our published methods in UQ, Bayesian inverse problems, sensitivity analysis, and physics-informed machine learning available through *QUEENS* while striving for applicability for large-scale numerical problems. We invite other interested groups and researchers to use and actively contribute to *QUEENS* in our repository, which can be found under <https://github.com/queens-py/queens>, along with complete documentation of the code and continuous integration pipelines.

If you want to use *QUEENS* in your published research, it is mandatory, given the generally accepted guidelines for good scientific practice, to cite this *QUEENS* software paper.

Notes on origin and evolution of *QUEENS* The software project *QUEENS* was initiated in 2016 by AdCo Engineering^{GW} GmbH (abbreviated as “AdCo” in the following). From 2017 through 2020, with JB being the lead developer at AdCo, it was supported by the Bavarian Ministry of Economic Affairs, Regional Development and Energy via their program BayTOU for promoting technology-oriented start-ups, as will be detailed in the subsequent Acknowledgements. In 2019, a collaboration with the Institute for Computational Mechanics of the Technical University of Munich (TUM) was started, aiming at further development of the software with respect to new innovative methods for uncertainty quantification, optimization, inverse analysis, etc. Later on, on the one hand, *QUEENS* was made available by AdCo to the TUM Institute for Computational Mechanics for redesign and further expansion (meanwhile as an Open Source code together with the Institute for Mathematics and Computer-Based Simulation of the University of the Bundeswehr Munich). On the other hand, starting from *QUEENS*, the commercial software SQUEENS (“Software for Quantifying Uncertain Effects in Engineering Systems”) was developed and established, among others, enabling the use of widely used commercial CAE software packages (such as ANSYS) as well as open-source CAE software packages (such as OpenFOAM) within SQUEENS and featuring a convenient graphical user interface (GUI). On 16 November 2021, SQUEENS was registered as a trademark and has been continuously supported by AdCo to date.

Acknowledgements We gratefully acknowledge the initial funding for *QUEENS*, received by AdCo Engineering^{GW} GmbH within the scope of the BayTOU project *QUEENS* (project no TOU-1606-0003) supported by the Bavarian Ministry of Economic Affairs, Regional Development and Energy through its program BayTOU. We furthermore want to thank the following funding sources that allowed us to continue the work on *QUEENS*: JN acknowledges support from the German Research Foundation (DFG) within the priority program SPP 1886. Furthermore, WAW, MD, and JN acknowledge support from the European Research Council (grant agreement No. 101021526-BREATHE). GRR acknowledges support from the German Federal Ministry of Education and Research (FestBatt 2 03XP0435B). Finally, we want to thank our former students and collaborators.

Author Contributions Jonas Biehler: Conceptualization, Methodology, Software, Validation, Supervision, Writing - Review and Editing, Project administration, Funding acquisition. Jonas Nitzler: Conceptualization, Methodology, Software, Validation, Formal analysis, Investigation, Writing - Original Draft, Writing - Review and Editing, Visualization, Project administration. Sebastian Brandstaeter: Conceptualization, Methodology, Software, Validation, Writing - Original Draft, Writing - Review and Editing, Project administration. Maximilian Dinkel: Methodology, Software, Validation, Writing - Original Draft, Writing - Review and Editing, Project administration. Volker Gravemeier: Conceptualization, Software, Funding acquisition. Lea J. Haeusel: Methodology, Software, Validation, Writing - Review and Editing, Project administration. Gil Robalo Rei: Methodology, Software, Validation, Writing - Original Draft, Writing - Review and Editing, Project administration. Harald Willmann: Methodology, Software, Validation, Formal analysis, Investigation, Project administration. Barbara Wirthl: Methodology, Software, Validation, Formal analysis, Investigation, Writing - Original Draft, Writing - Review and Editing, Visualization, Project administration. Wolfgang A. Wall: Conceptualization, Resources, Supervision, Writing - Review and Editing, Project administration, Funding acquisition.

Funding Open Access funding enabled and organized by Projekt DEAL.

Data availability The software described in this article (QUEENS) is developed openly at <https://github.com/queens-py/queens>. All other data that support the findings of this study are available from the corresponding author upon reasonable request.

Declarations

Conflict of interest JB and VG are employed by AdCo Engineering^{GW} GmbH. VG and WAW are shareholders of AdCo Engineering^{GW} GmbH. All other authors declare that they have no conflict of interest.

Open Access This article is licensed under a Creative Commons Attribution 4.0 International License, which permits use, sharing, adaptation, distribution and reproduction in any medium or format, as long as you give appropriate credit to the original author(s) and the source, provide a link to the Creative Commons licence, and indicate if changes were made. The images or other third party material in this article are included in the article's Creative Commons licence, unless indicated otherwise in a credit line to the material. If material is not included in the article's Creative Commons licence and your intended use is not permitted by statutory regulation or exceeds the permitted use, you will need to obtain permission directly from the copyright holder. To view a copy of this licence, visit <http://creativecommons.org/licenses/by/4.0/>.

References

- Baudin M, Christopoulou M, Collette Y, Martinez J-M (2013) pyDOE: the experimental design package for python. <https://pythonhosted.org/pyDOE/>. Accessed: 2025-10-07
- Iwanaga T, Usher W, Herman J (2022) Toward SALib 2.0: advancing the accessibility and interpretability of global sensitivity analyses. *Soc Environ Syst Model* 4:18155. <https://doi.org/10.18174/sesmo.18155>
- Feinberg J, Langtangen HP (2015) Chaospy: an open source tool for designing methods of uncertainty quantification. *J Comput Sci* 11:46–57. <https://doi.org/10.1016/j.jocs.2015.08.008>
- Marelli S, Sudret B (2014) UQLab: a framework for uncertainty quantification in Matlab. In: *Vulnerability, Uncertainty, and Risk*, American Society of Civil Engineers, pp. 2554–2563. <https://doi.org/10.1061/9780784413609.257>
- Olivier A, Giovanis DG, Aakash BS, Chauhan M, Vandanapu L, Shields MD (2020) UQpy: a general purpose python package and development environment for uncertainty quantification. *J Comput Sci* 47:101204. <https://doi.org/10.1016/j.jocs.2020.101204>
- Riis NAB, Alghamdi AMA, Uribe F, Christensen SL, Afkham BM, Hansen PC, Jørgensen JS (2024) Cuqipy: I. Computational uncertainty quantification for inverse problems in python. *Inverse Prob* 40(4): 045009. <https://doi.org/10.1088/1361-6420/ad22e7>
- Alghamdi AMA, Riis NAB, Afkham BM, Uribe F, Christensen SL, Christian Hansen P, Jørgensen JS (2024) Cuqipy: II. Computational uncertainty quantification for PDE-based inverse problems in Python, *Inverse Prob* 40(4):045010. <https://doi.org/10.1088/1361-6420/ad22e8>
- Lintusaari J, Vuollekoski H, Kangasrääsiö A, Skytén K, Järvenpää M, Marttinen P, Gutmann MU, Vehtari A, Corander J, Kaski S (2018) Elfi: engine for likelihood-free inference. *J Mach Learn Res* 19(16):1–7. <http://jmlr.org/papers/v19/17-374.html>
- Oriol A-P, Virgile A, Colin C, Larry D, F C J, Maxim K, Ravin K, Jupeng L, L CC, M OA, Michael O, Ricardo V, Thomas W, Robert Z (2023) Pymc: a modern and comprehensive probabilistic programming framework in python. *PeerJ Comput Sci* 9:e1516. <https://doi.org/10.7717/peerj-cs.1516>
- Chopin N, Papaspiliopoulos O (2020) An introduction to sequential monte carlo. Springer Int Publ. <https://doi.org/10.1007/978-3-030-47845-2>
- Seelinger L, Cheng-Seelinger V, Davis A, Parno M, Reinartz A (2023) UM-Bridge: Uncertainty quantification and modeling bridge. *J Open Sour Softw* 8(83):4748. <https://doi.org/10.21105/joss.04748>
- Adams BM, Bohnhoff WJ, Dalbey KR, Ebeida MS, Eddy JP, Eldred MS, Hooper RW, Hough PD, Hu KT, Jakeman JD, et al. (2021) Dakota, a multilevel parallel object-oriented framework for design optimization, parameter estimation, uncertainty quantification, and sensitivity analysis (v.6.16 user's manual). In: *Tech. rep., Sandia National Laboratories (SNL-NM), Albuquerque, NM (United States)*. <https://doi.org/10.2172/1868142>
- Baudin M, Dutfoy A, Iooss B, Popelin A-L (2017) OpenTURNS: an industrial software for uncertainty quantification in simulation. In: *Ghanem R, Higdon D, Owhadi H (eds) Handbook of Uncertainty Quantification*, Springer International Publishing, Cham, pp. 2001–2038. https://doi.org/10.1007/978-3-319-12385-1_64
- Richardson RA, Wright DW, Edeling W, Jancauskas V, Lakhili J, Coveney PV (2020) Easyvuvq: a library for verification, validation and uncertainty quantification in high performance computing. *J Open Res Softw*. <https://doi.org/10.5334/jors.303>
- Kroese DP, Taimre T, Botev ZI (2011) Handbook of monte carlo methods. In: 1st Edition, Wiley Series in Probability and Statistics, vol. 20. Wiley, Hoboken. <https://doi.org/10.1002/9781118014967>
- McKay MD, Beckman RJ, Conover WJ (1979) A comparison of three methods for selecting values of input variables in the analysis of output from a computer code. *Technometrics* 21(2):239–245. <https://doi.org/10.2307/1268522>
- Sobol' I (1967) On the distribution of points in a cube and the approximate evaluation of integrals. *USSR Comput Math Math Phys* 7(4):86–112. [https://doi.org/10.1016/0041-5553\(67\)90144-9](https://doi.org/10.1016/0041-5553(67)90144-9)

18. Levenberg K (1944) A method for the solution of certain nonlinear problems in least squares. *Q Appl Math* 2(2):164–168. <https://doi.org/10.1090/qam/10666>
19. Marquardt DW (1963) An algorithm for least-squares estimation of nonlinear parameters. *J Soc Ind Appl Math* 11(2):431–441. <https://doi.org/10.1137/0111030>
20. Virtanen P, Gommers R, Oliphant TE, Haberland M, Reddy T, Cournapeau D, Burovski E, Peterson P, Weckesser W, Bright J, van der Walt SJ, Brett M, Wilson J, Millman KJ, Mayorov N, Nelson ARJ, Jones E, Kern R, Larson E, Carey CJ, Polat İ, Feng Y, Moore EW, VanderPlas J, Laxalde D, Perktold J, Cimrman R, Henriksen I, Quintero EA, Harris CR, Archibald AM, Ribeiro AH, Pedregosa F, van Mulbregt P, SciPy 1.0 Contributors (2020) SciPy 1.0: fundamental algorithms for scientific computing in python. *Nat Methods* 17:261–272. <https://doi.org/10.1038/s41592-019-0686-2>
21. Kingma DP, Ba J (2017) Adam: a method for stochastic optimization. [arXiv:1412.6980](https://arxiv.org/abs/1412.6980)
22. Tieleman T, Hinton G, et al (2012) Lecture 6.5-rmsprop: divide the gradient by a running average of its recent magnitude. COURSE: Neural Netw Mach Learn 4(2):26–31
23. Nitzler J, Biehler J, Fehn N, Koutsourelakis P-S, Wall WA (2022) A generalized probabilistic learning approach for multi-fidelity uncertainty quantification in complex physical simulations. *Comput Methods Appl Mech Eng* 400:115600. <https://doi.org/10.1016/j.cma.2022.115600>
24. Biehler J, Mäck M, Nitzler J, Hanss M, Koutsourelakis P-S, Wall WA (2019) Multifidelity approaches for uncertainty quantification. *GAMM-Mitteilungen* 42(2): e201900008. <https://doi.org/10.1002/gamm.201900008>
25. Biehler J, Gee MW, Wall WA (2015) Towards efficient uncertainty quantification in complex and large-scale biomechanical problems based on a bayesian multi-fidelity scheme. *Biomech Model Mechanobiol* 14:489–513. <https://doi.org/10.1007/s10237-014-0618-0>
26. Koutsourelakis P-S (2009) Accurate uncertainty quantification using inaccurate computational models. *SIAM J Sci Comput* 31(5):3274–3300. <https://doi.org/10.1137/080733565>
27. Morris MD (1991) Factorial sampling plans for preliminary computational experiments. *Technometrics* 33(2): 161–174. <https://doi.org/10.1080/00401706.1991.10484804>
28. Sobol IM (1993) Sensitivity estimates for nonlinear mathematical models. *Math Modell Comput Exp* 1(4):407–414
29. Sobol IM (2001) Global sensitivity indices for nonlinear mathematical models and their Monte Carlo estimates. *Math Comput Simul* 55(1):271–280. [https://doi.org/10.1016/S0378-4754\(00\)00270-6](https://doi.org/10.1016/S0378-4754(00)00270-6)
30. Le Gratiet L, Cannamela C, Iooss B (2014) A bayesian approach for global sensitivity analysis of (multifidelity) computer codes. *SIAM/ASA J Uncert Quantif* 2(1):336–363. <https://doi.org/10.1137/130926869>
31. Wirthl B, Brandstaeter S, Nitzler J, Schrefler BA, Wall WA (2023) Global sensitivity analysis based on Gaussian-process metamodeling for complex biomechanical problems. *Int J Numer Methods Biomed Eng* 39(3):e3675. <https://doi.org/10.1002/cnm.3675>
32. Rasmussen CE, Williams CKI (2006) Gaussian processes for machine learning. The MIT Press, Cambridge, London
33. Hensman J, Fusi N, Lawrence ND (2013) Gaussian processes for big data. In: *Proceedings of the Twenty-Ninth Conference on Uncertainty in Artificial Intelligence (UAI2013)* 1309.6835, pp. 282–290
34. MacKay DJ (1995) Bayesian neural networks and density networks. *Nucl Instrum Methods Phys Res Sect A Accelerat Spectrom Detect Assoc Equip* 354(1):73–80. [https://doi.org/10.1016/0168-9002\(94\)00931-7](https://doi.org/10.1016/0168-9002(94)00931-7)
35. Hoffman MD, Gelman A (2014) The no-u-turn sampler: adaptively setting path lengths in hamiltonian monte carlo. *J Mach Learn Res* 15(47): 1593–1623. <https://jmlr.org/papers/v15/hoffman14a.html>
36. Del Moral P, Doucet A, Jasra A (2006) Sequential Monte Carlo samplers. *J R Stat Soc Ser B (Stat Methodol)* 68(3):411–436. <https://doi.org/10.1111/j.1467-9868.2006.00553.x>
37. Mohamed S, Rosca M, Figurnov M, Mnih A (2020) Monte carlo gradient estimation in machine learning. *J Mach Learn Res* 21(132):1–62. <https://jmlr.org/papers/v21/19-346.html>
38. Blei DM, Kucukelbir A, McAuliffe JD (2017) Variational inference: a review for statisticians. *J Am Stat Assoc* 112(518): 859–877. <https://doi.org/10.1080/01621459.2017.1285773>
39. Hoffman MD, Blei DM, Wang C, Paisley J (2013) Stochastic variational inference. *J Mach Learn Res* 14(40): 1303–1347. <https://jmlr.org/papers/v14/hoffman13a.html>
40. Kingma DP, Salimans T, Welling M (2015) Variational Dropout and the Local Reparameterization Trick. In: Cortes C, Lawrence N, Lee D, Sugiyama M, Garnett R (eds) *Advances in neural information processing systems*, vol 28. Curran Associates Inc, pp 2575–2583
41. Roeder G, Wu Y, Duvenaud DK (2017) Sticking the landing: simple, lower-variance gradient estimators for variational inference. In: Guyon I, Luxburg UV, Bengio S, Wallach H, Fergus R, Vishwanathan S, Garnett R (eds) *Advances in neural information processing systems*, vol 30. Curran Associates Inc, pp 6926–6935
42. Ranganath R, Gerrish S, Blei D (2014) Black box variational inference. In: *Artificial intelligence and statistics*, PMLR, pp. 814–822
43. Arenz O, Neumann G, Zhong M (2018) Efficient gradient-free variational inference using policy search. In: Dy J, Krause A (eds) *Proceedings of the 35th International Conference on Machine Learning*, Vol. 80 of *Proceedings of Machine Learning Research*, PMLR, pp. 234–243
44. Nitzler J, Temür BZ, Koutsourelakis P-S, Wall WA (2026) Efficient bayesian multi-fidelity inverse analysis for expensive and non-differentiable physics-based simulations in high stochastic dimensions. *Comput Methods Appl Mech Eng* 448:118442. <https://doi.org/10.1016/j.cma.2025.118442>
45. Dinkel M, Geitner CM, Rei GR, Nitzler J, Wall WA (2024) Solving bayesian inverse problems with expensive likelihoods using constrained gaussian processes and active learning. *Inverse Prob* 40(9):095008. <https://doi.org/10.1088/1361-6420/ad5eb4>
46. Saltelli A (ed) (2004) Sensitivity analysis in practice: a guide to assessing scientific models. Wiley, Hoboken, NJ
47. Brandstaeter S, Fuchs SL, Biehler J, Aydin RC, Wall WA, Cyron CJ (2021) Global sensitivity analysis of a homogenized constrained mixture model of arterial growth and remodeling. *J Elast* 145(1–2):191–221. <https://doi.org/10.1007/s10659-021-09833-9>
48. Nitzler J, Meier C, Müller KW, Wall WA, Hodge NE (2021) A novel physics-based and data-supported microstructure model for part-scale simulation of laser powder bed fusion of ti-6al-4v. *Adv Model Simul Eng Sci* 8(1):16. <https://doi.org/10.1186/s40323-021-00201-9>
49. Meier C, Fuchs SL, Much N, Nitzler J, Penny RW, Praegla PM, Proell SD, Sun Y, Weissbach R, Schreter M, Hodge NE, John Hart A, Wall WA (2021) Physics-based modeling and predictive simulation of powder bed fusion additive manufacturing across length scales. *GAMM-Mitteilungen* 44(3): e202100014. <https://onlinelibrary.wiley.com/doi/pdf/10.1002/gamm.202100014>
50. Willmann H, Wall WA (2022) Inverse analysis of material parameters in coupled multi-physics biofilm models. *Adv Model Simul Eng Sci* 9(1):1–32. <https://doi.org/10.1186/s40323-022-00220-0>
51. Kremheller J, Brandstaeter S, Schrefler BA, Wall WA (2021) Validation and parameter optimization of a hybrid embedded/

- homogenized solid tumor perfusion model. *Int J Numer Methods Biomed Eng* 37(8):e3508. <https://doi.org/10.1002/cnm.3508>
52. Willmann H, Nitzler J, Brandstätter S, Wall WA (2022) Bayesian calibration of coupled computational mechanics models under uncertainty based on interface deformation. *Adv Model Simul Eng Sci* 9(1):24. <https://doi.org/10.1186/s40323-022-00237-5>
 53. Hervas-Raluy S, Wirthl B, Guerrero PE, Robalo Rei G, Nitzler J, Coronado E, de Mora Sainz JF, Schrefler BA, Gomez-Benito MJ, García-Aznar JM, Wall WA (2023) Tumour growth: An approach to calibrate parameters of a multiphase porous media model based on in vitro observations of neuroblastoma spheroid growth in a hydrogel microenvironment. *Comput Biol Med* 159:106895. <https://doi.org/10.1016/j.combiomed.2023.106895>
 54. Dask Core Developers (2016) Dask: Library for dynamic task scheduling. <https://dask.org>. Accessed: 2025-10-07
 55. Feng H, Misra V, Rubenstein D (2007) PBS: A unified priority-based scheduler. *SIGMETRICS Perform Eval Rev* 35(1):203–214. <https://doi.org/10.1145/1269899.1254906>
 56. Yoo AB, Jette MA, Grondona M (2003) SLURM: Simple linux utility for resource management. In: Feitelson D, Rudolph L, Schwiegelshohn U (eds) *Job scheduling strategies for parallel processing*. Springer, Berlin Heidelberg, pp 44–60
 57. Haeusel LJ, Nitzler J, Köglmeier LJ, Wall WA (2026) Multi-physics-enhanced Bayesian inverse analysis: Information gain from additional fields. *Comput Methods Appl Mech Eng* 452:118735. <https://doi.org/10.1016/j.cma.2026.118735>

Publisher's Note Springer Nature remains neutral with regard to jurisdictional claims in published maps and institutional affiliations.

Authors and Affiliations

Jonas Biehler^{1,3}  · Jonas Nitzler¹  · Sebastian Brandstaeter²  · Maximilian Dinkel¹  · Volker Gravemeier³  · Lea J. Haeusel¹  · Gil Robalo Rei¹  · Harald Willmann¹  · Barbara Wirthl¹  · Wolfgang A. Wall¹ 

✉ Sebastian Brandstaeter
sebastian.brandstaeter@unibw.de

Jonas Biehler
jonas.biehler@tum.de

Jonas Nitzler
jonas.nitzler@tum.de

Maximilian Dinkel
maximilian.dinkel@tum.de

Volker Gravemeier
gravemeier@adco-engineering-gw.com

Lea J. Haeusel
lea.haeusel@tum.de

Gil Robalo Rei
gil.rei@tum.de

Harald Willmann
harald.willmann@tum.de

Barbara Wirthl
barbara.wirthl@tum.de

Wolfgang A. Wall
wolfgang.a.wall@tum.de

¹ Institute for Computational Mechanics, Technical University of Munich, Boltzmannstr. 15, 85748 Garching, Germany

² Institute for Mathematics and Computer-Based Simulation, University of the Bundeswehr Munich, Werner-Heisenberg-Weg 39, 85577 Neubiberg, Germany

³ AdCo Engineering GW GmbH, Oskar-Messter-Str. 33, 85737 Ismaning, Germany