INOVESALIB: A MODULAR VLASOV-FOKKER-PLANCK SOLVER FRAMEWORK

F. Donoso*, M. Brosi, E. Bründermann, M. Frank, S. Funkner, J. Gethmann,
A.-S. Müller, P. Schreiber,
Karlsruhe Institute of Technology, Karlsruhe, Germany

Abstract

The numerical solution of the Vlasov–Fokker–Planck (VFP) equation is a well-established method to simulate the dynamics of electron bunches in storage rings, including their self-interaction through wake fields. Inovesa is an efficient VFP solver that enables accurate simulations of phase space evolution, capturing phenomena such as coherent synchrotron radiation (CSR) and the micro-bunching instability on standard desktop hardware.

Building on this foundation, we introduce Inovesalib, a redesigned and extensible library version of Inovesa that exposes the core VFP solver as a modular C++ API and provides a Python wrapper for seamless integration into user workflows. Inovesalib decouples the solver core from application logic and offers a plugin architecture that allows users to implement custom algorithms without modifying the underlying solver. The redesigned structure turns Inovesa from a standalone application into a flexible simulation framework, suitable for integration into optimization pipelines, machine learning workflows, and custom tools. We present the library architecture and demonstrate its use in both C++ and Python.

INTRODUCTION

The dynamics of electron bunches in storage rings are governed by complex interactions between the beam and its electromagnetic environment. The phase space density evolution of an electron bunch is of particular relevance for understanding collective effects caused by perturbations through wake fields [1], or equivalently, through the beam coupling impedance. These interactions give rise to phenomena such as coherent synchrotron radiation (CSR) and micro-bunching instabilities, which can significantly degrade beam quality, reduce beam lifetime, and limit the performance of synchrotron radiation facilities [2, 3]. Specifically, the longitudinal dynamics can be described by the Vlasov-Fokker-Planck equation (VFPE), given in Eq. (1), which models the particle density of a bunch in the phase space and in time [3, 4].

$$\frac{\partial\psi}{\partial t} + \frac{\partial H}{\partial p}\frac{\partial\psi}{\partial q} - \frac{\partial H}{\partial q}\frac{\partial\psi}{\partial p} = \beta_d \frac{\partial}{\partial p}(p\psi) + D\frac{\partial^2\psi}{\partial p^2}. \quad (1)$$

Here, ψ denotes the phase space density, H is the Hamiltonian of the system, q and p are the canonical position and momentum coordinates, respectively, and β_d and D represent the damping and diffusion coefficients.

* felipe.donoso@kit.edu

To solve this equation efficiently in practice, Inovesa was developed as a runtime-efficient VFPE solver for desktop-class hardware. The original code base, developed by Schönfeldt at KIT [5], combined a discretized longitudinal phase space model with optimized numerical operators and parallelization, making systematic micro-bunching studies practical on standard workstations within minutes rather than days.

Since its introduction, Inovesa has been applied in a wide range of scientific studies. Steinmann *et al.* [6] used Inovesa to compare time-resolved THz spectroscopic measurements at KARA with VFP simulations, thereby relating measured bursting dynamics to simulated longitudinal substructures and threshold behavior. Schönfeldt *et al.* [7] used Inovesa to investigate how the CSR impedance contributes to the synchronous phase shift at KARA, connecting the solver to an experimentally accessible machine observable. In a related study, Funkner *et al.* [8] developed a non-destructive longitudinal phase space density tomography method, qualitatively validating their reconstructed dynamics against Inovesa simulations driven by KARA machine parameters.

Inovesa has also been used to interpret parameter-dependent instability studies. Brosi *et al.* [9] analyzed the effect of a damping wiggler on the micro-bunching instability at KARA and used Inovesa simulations to study how changes in damping time modify the temporal burst pattern of the emitted CSR. Boltz *et al.* [10] employed Inovesa for passive particle tracking in order to study how CSR wake fields perturb synchrotron motion and may seed micro-structures in longitudinal phase space. Expanding its application, Brosi et al. [11] utilized Inovesa as a primary VFP solver in a cross-code comparison between SOLEIL and KARA. Most recently, Brosi *et al.* [12] applied it at MAX IV to investigate microwave instabilities under varying intra-beam scattering (IBS) conditions, finding good qualitative agreement between simulation and experiment. These studies highlight the role of Inovesa as a reference tool for comparative and interpretive numerical studies, extending its use beyond individual simulation cases.

More recent work used Inovesa for hardware-oriented design, impedance manipulation, and control-oriented machine learning studies. Maier *et al.* [13] simulated the impact of corrugated structures on the longitudinal beam dynamics and emitted CSR power at KARA, and subsequently studied how these effects depend on bunch length conditions in low momentum compaction operation [14]. Wang *et al.* [15] used Inovesa as the CSR simulation environment for accelerated deep reinforcement learning aimed at fast feedback

of beam dynamics at KARA, showing that Inovesa can also serve as a simulation backend for low latency control system development.

The original Inovesa software was primarily organized as an application, which is suitable for batch runs and command-line studies but restrictive when the solver must be embedded into other software. Modern use cases such as optimization loops, machine learning workflows, online analysis tools, and custom graphical interfaces require programmatic access while allowing configuration, data export, runtime control, and alternative numerical realizations to evolve independently.

Inovesalib addresses this transition from application to framework. It preserves the validated Inovesa solver path [16], but reorganizes the software into a modular library with explicit interfaces for models, observers, controllers, and algorithms via plugin loading. In practical terms, the same physical problem can now be solved through a compact C++ API, dynamically loaded algorithm implementations, or Python bindings, making the solver easier to integrate, extend, and maintain.

ARCHITECTURE

Inovesalib is organized as a modular scientific software framework built around the validated numerical foundations of Inovesa. Its architecture separates solver execution, parameter models, diagnostics, runtime control, and algorithmic extension into distinct layers, ensuring that each component can evolve independently while remaining connected through stable interfaces.

Inovesa

At the core of Inovesalib lies the existing Inovesa VFP solver, now encapsulated in the `Inovesa class` as a reusable library component. This class follows a facade pattern that preserves the validated Inovesa numerical workflow while exposing it through a compact object interface for external applications, analysis pipelines, and Python-based workflows. In this way, the established solver kernel is reorganized into a form with explicit and reusable state management, runtime control, and data access.

The time evolution is carried out in the main iteration loop within the Inovesa class. For each step, the wake potential is updated, and the transport is performed by sequentially applying the numerical operators to the phase space state in the order of wake, RF, drift, and Fokker-Planck [16, 17]. This realizes the operator-splitting scheme in a form that remains close to the validated implementation while making the solver accessible through a stable library API. In addition, the Inovesalib supports selective operator refresh during runtime, so that controller-driven parameter changes can update the affected numerical maps without rebuilding the full solver instance.

Parameter models

Inovesalib introduces an explicit parameter model layer that separates simulation parameter configuration from solver execution. In this software context, a model is not the physical dynamics model itself, but a typed container for the physical, numerical, and execution parameters consumed by Inovesa and related algorithms. All parameter models derive from the abstract `ModelBase`, which provides a structured interface for describing, validating, and accessing simulation parameters at runtime through an explicit schema. This schema defines the stable interface through which a concrete parameter model interacts with the framework.

A key design feature is the separation between the parameter schema and the parameter values. Here, the schema denotes the metadata that describes the available parameters, such as their names, mutability, units, grouping, and value types. The method `describeAttributes()` publishes this schema, whereas the actual parameter values remain stored in the concrete model instance and are accessed through `getAttribute()` and `setAttribute()`. This separation provides a stable interface for controllers, observers, algorithms, and Python bindings, while leaving each parameter model free to choose its internal representation. In this way, parameter models define a clear boundary between parameter specification and solver implementation.

Observers

Output generation and diagnostics are decoupled from the numerical solver through an observer pattern. All observer implementations derive from the common `ObserverBase`, which provides the minimal polymorphic interface used to attach output-side components to the solver. This separation prevents simulation code from being entangled with file I/O, visualization, benchmarking, or communication back-ends, and makes it possible to combine several output pathways with the same solver run.

Several observer families are already implemented, including `InovesaLegacy` for asynchronous HDF5 [18] output, ZeroMQ [19] for streaming phase space and projection arrays, and additional observer implementations for plot generation and metadata reporting. Together, they allow the same physics core to be reused in batch simulations, interactive applications, and performance studies without modifying the solver logic.

Controllers

Complementary to the observer layer, controllers provide a mechanism for influencing the solver during runtime. Their purpose is not to consume output, but to modify the model or trigger solver updates while the simulation is running. This is particularly relevant for interactive steering, parameter scans with feedback, and future hardware-oriented or online applications.

The main controller currently implemented in Inovesalib is the ZMQ model control. It receives JSON-encoded commands, validates requested changes against the model

schema, and applies accepted updates during iteration. These changes can selectively refresh the RF, drift, Fokker-Planck, or wake operators, depending on which parameters have been modified. This architecture keeps runtime steering outside the numerical core while preserving consistency between the model state and the active operator chain.

Algorithm Plugins

InovesaLib adopts a plugin-based architecture to organize families of solver algorithms. For the Inovesa VFP solver, this is expressed through the `APIVFPAlgorithm` interface, which defines the common contract for all VFP solver strategies. In this way, multiple VFP algorithms can address the same problem through a shared interface. This makes the framework inherently extensible: the core solver is treated merely as a baseline implementation. Researchers can optionally develop plugins to introduce entirely new types of algorithms, such as novel numerical schemes, data-driven methods, or alternative physics models, expanding the library's capabilities without ever modifying the core codebase.

This design follows the strategy pattern: at runtime, the framework selects and instantiates a concrete algorithm. These algorithms are compiled as independent shared libraries and loaded dynamically, with each one registered under a unique string identifier. Consequently, researchers can seamlessly swap, benchmark, and deploy different mathematical approaches for a given physical problem without altering or recompiling the underlying framework infrastructure.

EXAMPLES

This section presents short code snippets illustrating the main usage patterns of Inovesalib in C++ and Python, showing how models, solvers, observers, and related components interact in typical workflows.

C++

The first example illustrates the high-level usage path, where an `Inovesa` instance is created from a configured model, observers and controllers are attached, and the simulation is executed. This reflects the most direct application-level workflow.

Listing 1: High-level workflow with `Inovesa`.

```
#include "inovesalib/Inovesalib.h"
int main() {
    inovesalib::InovesaModel param_model;
    inovesalib::Inovesa solver(param_model);
    inovesalib::LegacyObserver observer;
    inovesalib::ZmqModelController controller;
    solver.addObserver(&observer);
    solver.addController(&controller);
    solver.iterate(10000);
}
```

The second example shows the algorithm/strategy-oriented path. Here, the selected algorithm plugin is passed explicitly at construction time through `SolverVFP`, emphasizing how the execution strategy is chosen independently from model and observers.

Listing 2: Algorithm-oriented workflow with explicit strategy selection.

```
#include "inovesalib/Inovesalib.h"
int main() {
    inovesalib::InovesaModel param_model;
    vector<inovesalib::ObserverBase*> observers;
    vector<inovesalib::ControllerBase*> controllers;
    inovesalib::LegacyObserver observer;
    inovesalib::ZmqModelController controller;
    observers.push_back(observer);
    controllers.push_back(controller);
    string algorithm = "InovesaLegacy";
    inovesalib::SolverVFP solver(algorithm, observers,
     controllers, &param_model);
    solver.iterate(10000);
}
```

Python

The current version of the Python wrapper follows the same high-level idea as the C++ code: a parameter model is created, a solver instance is configured, observers are attached, and the simulation is executed. The snippet below shows a practical workflow.

Listing 3: Minimal Python example.

```
import inovesalib as inl

param_model = inl.model("InovesaLegacy")
param_model.grid_size = 256
param_model.beam_current = 8.5e-4
solver = inl.Inovesa(param_model)
observer = inl.InovesaLegacyObserver()
solver.addObserver(observer)
solver.iterate(10000)
```

CONCLUSION

Inovesalib extends the established Inovesa Vlasov–Fokker–Planck solver into a modular software framework while preserving its validated numerical core. By defining explicit interfaces for solver execution, parameter models, plugins, observers, controllers, and Python bindings, the same simulation engine can be reused beyond the original standalone application workflow.

The refactoring maintains continuity with previous physics studies while enabling new forms of integration through a compact user interface and a plugin mechanism for developers. Ongoing work focuses on a differentiable VFP solver that supports both forward simulation and gradient propagation for inverse problems and optimization workflows. Access to the Inovesalib codebase is available upon request to the corresponding author.

ACKNOWLEDGEMENTS

F.D. gratefully acknowledges support from the MathSEE PhD Bridge Program and the KIT Graduate School Computational and Data Science.

REFERENCES

[1] M. Dohlus and R. Wanzenberg, "An introduction to wake fields and impedances", *CERN Yellow Rep. School Proc.*, vol. 3, p. 15, 2017. doi:10.23730/CYRSP-2017-003.15

[2] M. Brosi, "Overview of the micro-bunching instability in electron storage rings and evolving diagnostics", in *Proc. IPAC'21*, Campinas, SP, Brazil, pp. 3686–3691, May 2021. doi:10.18429/JACoW-IPAC2021-THXA02

[3] M. Venturini, R. Warnock, R. Ruth, and J. A. Ellison, "Coherent synchrotron radiation and bunch stability in a compact storage ring", *Phys. Rev. Spec. Top. Accel. Beams*, vol. 8, no. 1, p. 014202, 2005. doi:10.1103/PhysRevSTAB.8.014202

[4] R. L. Warnock and J. A. Ellison, "A general method for propagation of the phase space distribution, with application to the sawtooth instability", Stanford Linear Accelerator Center, Stanford, CA, USA, Rep. SLAC-PUB-8404, 2000. doi:10.2172/753322

[5] P. Schönfeldt, M. Brosi, M. Schwarz, J. L. Steinmann, and A.-S. Müller, "Parallelized Vlasov–Fokker–Planck solver for desktop personal computers", *Phys. Rev. Accel. Beams*, vol. 20, no. 3, p. 030704, 2017. doi:10.1103/PhysRevAccelBeams.20.030704

[6] J. L. Steinmann *et al.*, "Continuous bunch-by-bunch spectroscopic investigation of the micro-bunching instability", *Phys. Rev. Accel. Beams*, vol. 21, no. 11, p. 110705, 2018. doi:10.1103/PhysRevAccelBeams.21.110705

[7] P. Schönfeldt *et al.*, "Study of the influence of the CSR impedance on the synchronous phase shift at KARA", in *Proc. IPAC'18*, Vancouver, BC, Canada, pp. 2223–2226, Apr. 2018. doi:10.18429/JACoW-IPAC2018-WEPAL028

[8] S. Funkner *et al.*, "Revealing the dynamics of ultrarelativistic non-equilibrium many-electron systems with phase space tomography", *Sci. Rep.*, vol. 13, no. 1, p. 4618, 2023. doi:10.1038/s41598-023-31196-5

[9] M. Brosi *et al.*, "Studies of the micro-bunching instability in the presence of a damping wiggler", *J. Phys.: Conf. Ser.*, vol. 1067, p. 062017, 2018. doi:10.1088/1742-6596/1067/6/062017

[10] T. Boltz *et al.*, "Perturbation of synchrotron motion in the micro-bunching instability", in *Proc. IPAC'19*, Melbourne, Australia, pp. 108–111, May 2019. doi:10.18429/JACoW-IPAC2019-MOPGW018

[11] M. Brosi *et al.*, "Simulations of the micro-bunching instability for SOLEIL and KARA using two different VFP solver codes", in *Proc. IPAC'22*, Bangkok, Thailand, pp. 2237–2240, Jun. 2022. doi:10.18429/JACoW-IPAC2022-WEPOMS005

[12] M. Brosi, F. Cullinan, J. Breunlin, and Åke. Andersson, "Investigation of the microwave instability at MAX IV laboratory in combination with intra-beam scattering", in *Proc. IPAC'25*, Taipei, Taiwan, pp. 2207–2210, Jun. 2025. doi:10.18429/JACoW-IPAC2025-WEPM099

[13] S. Maier *et al.*, "Simulation of the effect of corrugated structures on the longitudinal beam dynamics at KARA", in *Proc. IPAC'22*, Bangkok, Thailand, pp. 2241–2244, Jun. 2022. doi:10.18429/JACoW-IPAC2022-WEPOMS006

[14] S. Maier *et al.*, "Simulation studies on longitudinal beam dynamics manipulated by corrugated structures under different bunch length conditions at KARA", in *Proc. IPAC'23*, Venice, Italy, pp. 3570–3573, May 2023. doi:10.18429/JACoW-IPAC2023-WEPL189

[15] W. Wang *et al.*, "Accelerated deep reinforcement learning for fast feedback of beam dynamics at KARA", *IEEE Trans. Nucl. Sci.*, vol. 68, no. 8, pp. 1794–1800, 2021. doi:10.1109/TNS.2021.3084515

[16] P. Schönfeldt, "Simulation and measurement of the dynamics of ultra-short electron bunch profiles for the generation of coherent THz radiation", Ph.D. thesis, Karlsruhe Institute of Technology, Karlsruhe, Germany, 2018. doi:10.5445/IR/1000084466

[17] F. Donoso *et al.*, "Longitudinal phase space density tomography constrained by the Vlasov–Fokker–Planck equation", in *Proc. IPAC'24*, Nashville, TN, USA, pp. 2350–2353, May 2024. doi:10.18429/JACoW-IPAC2024-WEPG55

[18] The HDF Group and Q. Koziol, "HDF5-Version 1.12.0", Computer software, Feb. 2020, doi:10.11578/DC.20180330.1,

[19] P. Hintjens, *ZeroMQ*. Sebastopol, CA, USA: O'Reilly, 2013.