Software Impacts 17 (2023) 100563

Contents lists available at ScienceDirect

Software Impacts

journal homepage: www.journals.elsevier.com/software-impacts

Original software publication

Python Data Driven framework for acceleration of Phase-Field simulations

Seifallah Fetni *, Jocelyn Delahaye, Anne Marie Habraken

UEE Research Unit, University of Liège, Belgium

ARTICLE INFO

Keywords:
Python development
Deep learning
Image generation and processing
LSTM
PCA

ABSTRACT

The passage describes the development of a numerical framework in Python to create and process a large dataset for time-series prediction using Deep Learning algorithms. The dataset is generated by solving the Cahn–Hilliard equation for spinodal decomposition of a binary alloy and is labeled to train the algorithms. Prior to training, dimensionality reduction is performed using Auto-encoders and Principal Component Analysis. The framework identifies three distinct latent dimensions/spaces for the datasets. The primary dataset was generated by running up to 10,000 High-Fidelity Phase-Field simulations in parallel using High-Performance Computing (HPC). The framework is compatible with all major operating systems and has been thoroughly tested on Python 3.7 and later versions.

Code metadata

Current code version	*V1*
Permanent link to code/repository used for this code version	https://github.com/SoftwareImpacts/SIMPAC-2023-89
Permanent link to reproducible capsule	https://codeocean.com/capsule/7854717/tree/v1
Legal code license	*1 - This code is public under the Apache 2.0 permissive license.*
	2 - We warn users that we are using pyfftw
Code versioning system used	*gitlhub*
Software code languages, tools and services used	*Python*
Compilation requirements, operating environments and dependencies	–
If available, link to developer documentation/manual	
Support email for questions	seifallah.el.fetni@gmail.com

1. Motivation and significance

The combination of Phase-Field (PF) and Machine Learning (ML) has great potential for advancing scientific research in Materials Science [1]. However, one of the main challenges faced by researchers in this area is the lack of readily available data for training and testing ML models [2]. Python provides an open-source development environment that can be used to program PF simulations and generate rich datasets for ML and also to deal with close topics [3,4]. Thus, it could be possible to reproduce quantitative results and conduct advanced analyses.

However, despite the growing interest in PF and ML, there is still limited literature on programming PF simulations, especially in generating datasets for use in ML. As such, providing a complete example that addresses this challenge and generates PF datasets would be of great interest to the scientific community. Such a comprehensive example would lead to better understanding the complexities of PF simulations and the type of data they can generate, which could help spur innovation and accelerate the implementation of novel approaches.

The developed python framework and the generated datasets can be used to reproduce the work presented in the related article, and the framework can be extended to other research areas such as crack propagation in materials science or next-frame prediction. By making these datasets available to the scientific community, which could contribute to address some of the most pressing challenges in the associated research line.

The code (and data) in this article has been certified as Reproducible by Code Ocean: (https://codeocean.com/). More information on the Reproducibility Badge Initiative is available at https://www.elsevier.com/physical-sciences-and-engineering/computer-science/journals.

* Corresponding author.
E-mail address: seifallah.el.fetni@gmail.com (S. Fetni).

https://doi.org/10.1016/j.simpa.2023.100563
Received 9 March 2023; Received in revised form 27 July 2023; Accepted 1 August 2023

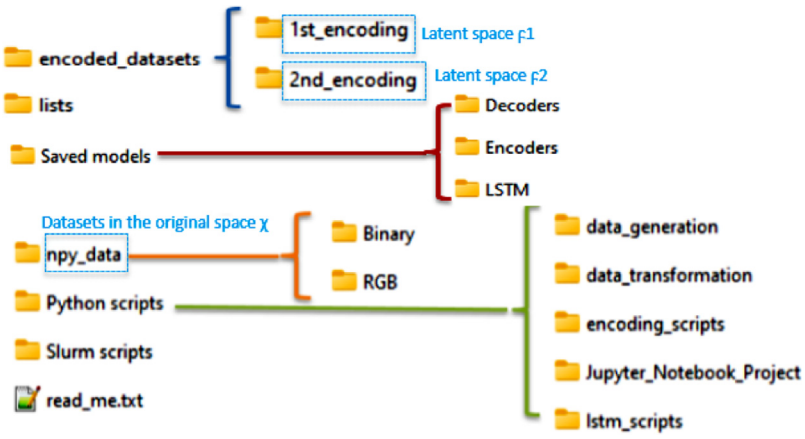

Fig. 1. Overview of the shared repository content: Data organization.

2. Software impacts

The developed framework has had a significant impact on the research area of phase-field simulations and machine learning. The framework provides researchers with a powerful tool to generate datasets from phase-field simulations, process the results, and train various deep learning algorithms. This has enabled the development of sophisticated models with improved accuracy and predictive capability. The framework has been used to get results for already published works, especially [5,6], demonstrating its relevance and usefulness, and ongoing ones. The dataset generated by the framework is based on high-fidelity phase-field simulations and has been shown to be suitable for training various deep learning algorithms, such as auto-encoders, Long Short Term Memory (LSTM), Gated Recurrent Units (GRU), and principal component analysis. The use of these algorithms has led to significant dimensionality reduction and time-series analysis. For example, the application of LSTM neural networks in [6] has shown the possibility of making next-frame predictions, allowing for the acceleration of phase-field simulations without the need for high computing resources. Moreover, the framework provides robust Python codes that allow for customization and easy integration with other tools. The provided scripts for running simulations under HPC environments are particularly useful for researchers dealing with large and complex datasets. In [5], various integration schemes to resolve phase-field problems were tested and compared, and then an adaptive stepping scheme is proposed to be a potential candidate to resolve advanced PF problems while defining the appropriate time step during the simulation. Additional works are now under consideration and development to deal with the correlation between additive manufacturing process parameters and the resulting mechanical properties of the built materials. The framework's impact is expected to grow even further as it can be extended to other research areas, such as crack propagation in materials science or next-frame prediction.

Overall, the developed framework has greatly facilitated the integration of PF simulations and ML and has led to new insights and advances in the field.

3. Framework description

3.1. Description of generation and the nature of the original dataset

3.1.1. Description of generation and the nature of the original dataset

This code allows first the generation of data related to the spinodal decomposition of a given binary alloy. It is governed by the Cahn–Hilliard equation [7] expressed as :

$$\frac{\partial X}{\partial t} = \nabla . M \ \nabla \frac{\partial G}{\partial X} \tag{1}$$

Here G denotes the total free energy of the binary alloy, M the mobility and X the molar fraction of an element of the alloy. If we neglect the terms related to the elastic strain, the total free energy G could be expressed as:

$$G(X) = \int \left\{ g(X) + \frac{1}{2}\kappa \ \nabla X^2 \right\} dV \tag{2}$$

where g is the chemical bulk energy and κ is here the gradient energy coefficient. Details about programming a single PF simulation is provided in [5].

The raw data is generated by running the python code in a Linux or Windows environment (tested in both). Fig. 1 shows the data organization as follows. The repository 'npy_data' contains a collection of datasets with different sizes (number of samples) or type (RGB or binary images). The one used to conduct the work in the associated research paper [6] is 'dataset_7000.npy', while the other datasets could be used for different purposes: training on a relatively small dataset, advanced analysis, trying other neural networks algorithms etc. The 'encoded_datasets' repository contains datasets in the latent spaces $\chi 1$ and $\chi 2$. A repository is dedicated to python scripts: the Jupyter Notebook project ('.ipynb') contains the whole numerical implementation of the framework (Figure 1 in [6]), while python files are associated with pre and post-processing tasks as described hereafter. For computing in HPC environments, some Simple Linux Utility for Resource Management (SLURM) scripts are provided to get an idea about the required computing resources. Similarly, some saved models after Deep Learning (DL) training are stored in the corresponding repository ('.h5' format).

Fig. 2 describes the content of the raw data folder ('sim_data') before transforming it into a '.npy' file for ease of download and loading. This description helps readers understand the data architecture, which is necessary for pre- and post-processing, as well as extracting simulation features for the time-series analysis phase (content of the folder 'lists'). Labeling data in this way is essential for efficient data management and analysis.

Fig. 3 provides a detailed description of the labeling process for each simulation. This labeling takes into account the concentration coefficient, the mobility coefficient, and the gradient coefficient, respectively. The equations associated with these coefficients are described in the Methods section.

3.1.2. Description of the encoded datasets (in two latent dimensions)

The repository '1st_encoding' in Fig. 1 contains the data encoded using the first auto-encoder. These data files have a '.txt' format. The associated name returns the number of samples and the size of the output code of the encoder respectively, to reduce the dimensionality of the data from the original space (χ) to the first latent space ($\mathbb{F}1$). As an example, 'encoded_data_4800_750.txt' means that this data contains 4800 samples, and it is obtained by applying an auto-encoder with

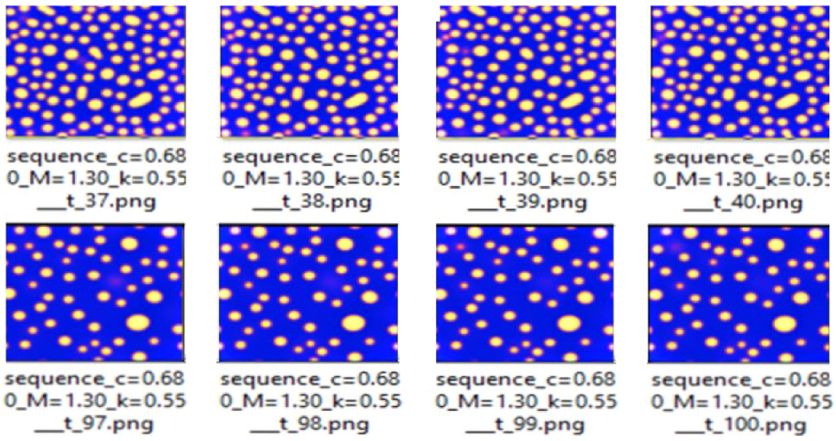

Fig. 2. This illustration shows the contents of the dataset. Each simulation is associated with its key parameters or variables, and each image corresponds to the state of the microstructure at a specific time step. Small details distinguish each frame from the others, and these details are important for Machine Learning to learn the evolution behavior.

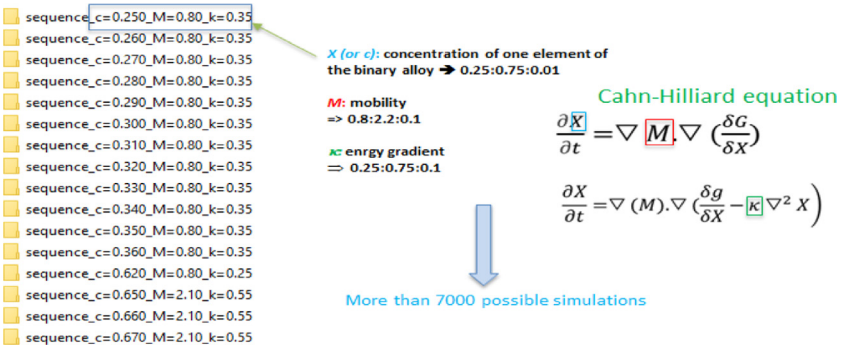

Fig. 3. Illustration of the data labeling process.

output code (750,) on the original dataset 'dataset_4800.npy'. Similarly, the dataset 'encoded_data_7000_750_250.txt' belongs to the second latent space (F2). It is obtained by applying two successive auto-encoders in a serial-way (or an auto-encoder followed by a PCA) on the original dataset 'dataset_4800.npy' with output code sizes (750,) and (250,) respectively.

3.2. Description of the main functions of the different Python scripts of the developed framework

It is worth noting that the entire framework in the Jupyter Notebook project '.ipynb' can be run by the user after loading the dataset in the first latent space (e.g. 'encoded_data_7000_750.txt'). However, it is recommended to split the '.ipynb' project into python scripts (.py) to enable serial execution of different phases of the framework in HPC environment. This can be achieved by submitting batch jobs for each script. For this purpose, we specify the main function, its inputs, and its outputs for each python script in Table 1. To perform post-processing of results obtained from each script or to perform sensitivity analysis, the user can upload the desired outputs to the Jupyter Notebook project on their local machine. This approach offers a flexible way to execute the numerical framework.

4. Limitations and future improvements

Despite the success achieved with the developed framework and generated datasets, there are still some limitations that could be addressed in future work. One possibility is to implement Variational Autoencoders (VAEs) as an alternative to standard Autoencoders for encoding phase-field simulation data. VAEs can provide significant dimensionality reduction while using smaller dataset sizes compared to traditional autoencoders. Additionally, the application of Denoising Autoencoders at the end of the pipeline can improve the reconstruction of the data, leading to better results in downstream ML applications. Implementing these techniques could result in even more efficient and accurate training of deep learning models for phase-field simulation data.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgments

Computational resources have been provided by the Consortium des Équipements de Calcul Intensif (CÉCI), funded by the Fonds de la Recherche Scientifique de Belgique (F.R.S.-FNRS), Belgium under Grant No. 2.5020.11 and by the Walloon Region, Belgium. A special thank to Mr. David Colignon for his availability and great support to successfully achieve computational tasks.

Ethics statements

This work did not involve human subjects, animal experiments, or data collected from social media platforms.

Table 1
Specification of the input(s) and output(s) of each python script.

Operation	Python script	Input	Output	Remarks
Data transformation	*data_to_npy.py*	Repository Containing all simulations	One single '.npy' file (transformed data)	We remind that the appellation '*dataset_N.npy*' means that the raw data contains *N* simulations
Generation of the simulations names List	*list_names.py*	Idem	A '.txt' file containing the names of all simulations	We remind that this file would be useful to extract simulations characteristics to ensure better training of the LSTM/GRU
Data verification	*load_data.py*	The raw data ('.npy') format	A sample image	To check that the '.npy' file was correctly generated during the previous operation
Auto-encoder 1 training	encoder_decoder_1.py	*The raw data ('. npy')*	Saved models (encoders and decoders), training and validation loss	A saved auto-encoder model ⇨ '*encoder_7000_750.h5*' means an encoder trained on the dataset 'dataset_7000.npy' with the output code size (750,)
Data encoding (1st encoder) X ➔ F1	*data_to_txt.py*	Transformed data + *Saved Auto-encoder 1*	Reduced dataset (latent dimension F1)	Example of input: '*dataset_4800.npy*' Example of output: '*encoded_data_4800_750.*txt'
Auto-encoder 2 training	encoder_decoder_2.py	Dataset in latent space (F1) ('.txt')	Saved models (encoders and decoders), training and validation loss	A saved auto-encoder model ⇨ '*encoder_7000_750_250.h5*' means an encoder trained on the dataset 'dataset_7000_750.txt with the output code size (250,)
Data encoding (2nd encoder) F1 ➔ F2	*data_to_txt_2.py*	Dataset in latent space (F1) + *Saved Auto-encoder 2*	Reduced dataset (latent dimension F2)	Example of output: '*encoded_data_4800_750_250.txt*'
LSTM/GRU Training	*'encoder_decoder_LSTM_2_enc_dec_layers.py'* *OR* *'encoder_decoder_LSTM_1_enc_dec_layer_PCA. py'*	Reduced dataset (latent dimension F2)	Saved models (LSTM/GRU), training and validation loss	Example of output: '*lstm_7000_750_pca_200.h5*' ➔ *an LSTM model correspondent to a dataset F2:* - Original dataset : 7000 samples - Encoder 1 reduction : (750,) - PCA reduction :200 components

References

[1] C. Hu, S. Martin, R. Dingreville, Accelerating phase-field predictions via recurrent neural networks learning the microstructure evolution in latent space, Comput. Methods Appl. Mech. Engrg. 397 (2022) 115128, http://dx.doi.org/10.1016/j.cma.2022.115128.

[2] K. Choudhary, et al., Recent advances and applications of deep learning methods in materials science, npj Comput. Mater. 8 (1) (2022) 59, http://dx.doi.org/10.1038/s41524-022-00734-6.

[3] A. Teurtrie, et al., Espm: A python library for the simulation of STEM-EDXS datasets, Ultramicroscopy 249 (2023) 113719, http://dx.doi.org/10.1016/j.ultramic.2023.113719.

[4] M.Y. Toriyama, J. Qu, L.C. Gomes, E. Ertekin, VTAnDeM: A python toolkit for simultaneously visualizing phase stability, defect energetics, and carrier concentrations of materials, Comput. Phys. Comm. 287 (2023) 108691, http://dx.doi.org/10.1016/j.cpc.2023.108691.

[5] S. Fetni, J. Delahaye, L. Duchêne, A. Mertens, A.M. Habraken, Adaptive time stepping approach forphase-field modeling of phase separation and precipitates coarsening in additive manufacturing alloys - COMPLAS 2021, in: COMPLAS 2021-16th Int. Conf. Comput. Plast. Fundam. Appl, 2021, pp. 1–12, http://dx.doi.org/10.23967/complas.2021.009.

[6] S. Fetni, et al., Capabilities of auto-encoders and principal component analysis of the reduction of microstructural images; application on the acceleration of phase-field simulations, Comput. Mater. Sci. 216 (2023) 111820, http://dx.doi.org/10.1016/j.commatsci.2022.111820.

[7] D. Montes de Oca Zapiain, J.A. Stewart, R. Dingreville, Accelerating phase-field-based microstructure evolution predictions via surrogate models trained by machine learning methods, npj Comput. Mater. 7 (1) (2021) http://dx.doi.org/10.1038/s41524-020-00471-8.