The Journal of Open Source Software

Sapsan: Framework for Supernovae Turbulence Modeling with Machine Learning

Platon I. Karpov[1, 2], **Iskandar Sitdikov**[3], **Chengkun Huang**[2], **and Chris L. Fryer**[2]

1 Department of Astronomy & Astrophysics, University of California, Santa Cruz, CA **2** Los Alamos National Laboratory, Los Alamos, NM **3** Provectus IT Inc., Palo Alto, CA

DOI: 10.21105/joss.03199

Software

- Review
- Repository
- Archive

Editor: Dan Foreman-Mackey

Reviewers:

- @kburns
- @milescranmer

Submitted: 02 April 2021
Published: 23 November 2021

Summary

Sapsan is a framework designed to make Machine Learning (ML) more accessible in the study of turbulence, with a focus on astrophysical applications. Sapsan includes modules to load, filter, subsample, batch, and split the data from hydrodynamic (HD) simulations for training and validation. Next, the framework includes built-in conventional and physically-motivated estimators that have been used for turbulence modeling. This ties into Sapsan's custom estimator module, aimed at designing a custom ML model layer-by-layer, which is the core benefit of using the framework. To share your custom model, every new project created via Sapsan comes with pre-filled, ready-for-release Docker files. Furthermore, training and evaluation modules come with Sapsan as well. The latter, among other features, includes the construction of power spectra and comparison to established analytical turbulence closure models, such as a gradient model. Thus, Sapsan attempts to minimize the hard work required for data preparation and analysis, leaving one to focus on the ML model design itself.

Statement of Need

Domain sciences have been slow to adopt Machine Learning (ML) for a range of projects, but particularly for physical simulations modeling turbulence. It is challenging to prove that an ML model has learned the laws of physics in a particular problem, and that it has the ability to extrapolate within the parameter-space of the simulation. The inability to directly infer the predictive capabilities of ML is one of the major causes behind the slow adoption rates; however, the community cannot ignore the effectiveness of ML.

Turbulence is ubiquitous in astrophysical environments, however, it involves physics at a vast range of temporal and spatial scales, making accurate fully-resolved modeling difficult. Various analytical turbulence models have been developed to be used in simulations using temporal or spatial averaged governing equations, such as RANS (Reynolds-averaged Navier-Stokes) and LES (Large Eddy Simulation), but the accuracy of these methods is sometimes inadequate. In search of better methods to model turbulence in core-collapse supernovae, it became apparent that ML has the potential to produce more accurate turbulence models on an unaveraged subgrid-scale than the current methods. Scientists from both industry and academia (King et al., 2016; Zhang et al., 2018) have already begun using ML for applied turbulent problems. Still, none of these efforts have yet reached the scales relevant for the physics and astronomy community on a practical level. For example, physics-based model evaluation and interpretability tools are not standardized, nor are they widely available. As a result, it is a common struggle to verify published results, with the setup not fully documented, the opaquely structured code lacking clear commenting, or even worse, not publicly available. This

is a problem that the broader ML community can relate to as well (Hutson, 2018). Thus, it is not surprising that there is considerable skepticism against ML in physical sciences, with astrophysics being no exception (Carleo et al., 2019).

In pursuit of our supernova (SNe) study, the issues outlined above became painfully apparent. Thus, we have attempted to lower the barrier to entry for new researchers in domain science fields studying turbulence to employ ML, with the main focus on astrophysical applications. As a result, we developed an ML Python-based pipeline called `Sapsan`. The goals have been to make this library accessible and shared with the community through Jupyter Notebooks, a command-line-interface (CLI) and a graphical-user-interface (GUI)[1] available for end-users. `Sapsan` includes built-in optimized ML models for turbulence treatment, both conventional and physics-based. More importantly, at its core, the framework is meant to be flexible and modular; hence there is an intuitive interface for users to work on their own ML algorithms. Most of the mundane turbulence ML researcher needs, such as data preprocessing and prediction analysis, can be automated through `Sapsan`, with a streamlined process of custom estimator development. In addition, `Sapsan` brings best practices from the industry regarding ML development frameworks. For example, `Sapsan` includes docker containers for reproducible release, as well as MLflow for experiment tracking. Thus, `Sapsan` is a single, complete interface for ML-based turbulence research.

`Sapsan` is distributed through GitHub and pip. For further reference, wiki is maintained on GitHub as well.

Framework

`Sapsan` organizes workflow via three respective stages: data preparation, machine learning, and analysis, as shown in Figure 1. The whole process can be further distributed using Docker for reproducibility. Let's break down each stage in the context of turbulence subgrid modeling, e.g., a model to predict turbulent behavior at the under-resolved simulation scales.

- **Data Preparation**
  - **Loading Data:** `Sapsan` is ready to process common 2D & 3D hydrodynamic (HD) and magnetohydrodynamic (MHD) turbulence data in simulation-code-specific data formats, such as HDF5 (with more to come per community need).
  - **Transformations:** A variety of tools are available for the user to prepare data for training:
    - **Filtering:** To build a subgrid model, one will have to filter the data to, for example, remove small-scale perturbations. Some possible choices include a box, spectral, or Gaussian filter. The data can be filtered on the fly within the framework.
    - **Sampling:** to run quick tests of your model, you might want to test on a sampled version of the data while retaining the full spatial domain. For this application, equidistant sampling is available in `Sapsan`.
    - **Batching & Splitting:** The data are spatially batched and divided into testing and validation subsets.
- **Machine Learning**
  - **Model Setup:** Different ML models may be appropriate for different physical regimes, and `Sapsan` provides templates for a selection of both conventional and physics-based models with more to come. Only the most important options are left up to the user to edit, with most overhead kept in the backend. This stage

[1] A demo is available at sapsan.app.

also includes tools for defining ML layers, tracking parameters, and choosing and tuning optimization algorithms.

- **Analysis**
  - **Trained Model:** A turbulence subgrid model defines how small-scale structure affects the large scale quantities. In other words, it completes or ''closes" the governing large-scale equations of motion with small-scale terms. The prediction from a trained ML model is used to provide the needed quantities.
  - **Analytical Tools:** There are also methods included for comparing the trained model with conventional analytic turbulence models [such as the Dynamic Smagorisnky, Lilly (1966); or Gradient, Liu et al. (1994); models], or to conduct other tests of, for example, the power spectrum of the model prediction.

For further information on each stage, please refer to Sapsan's Wiki on Gihub.

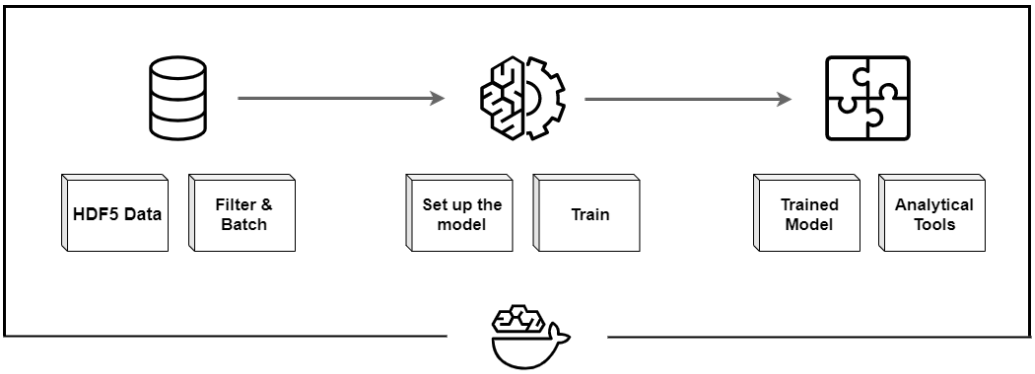

Figure 1: High-level overview of `Sapsan's` workflow.

Dependencies

The following is a list of the core functional dependencies[2] and a short description of how they are used within `Sapsan`:

- **PyTorch:** `Sapsan`, at large, relies on PyTorch to configure and train ML models. Thus, the parameters in the aforementioned **Model Setup** stage should be configured with PyTorch functions. Convolutional Neural Network (CNN) and Physics-Informed Convolutional Auto Encoder (PICAE) examples included with `Sapsan` are based on PyTorch. (Paszke et al., 2019)
- **Scikit-learn:** A alternative to `PyTorch`, as demonstrated in the Kernel Ridge Regression (KRR) example in `Sapsan`. Since `scikit-learn` is less flexible and scalable than `PyTorch`, `PyTorch` is the recommended interface. (Pedregosa et al., 2011)
- **Catalyst:** used as part of the backend to configure early-stopping of the model and logging. (Kolesnikov, 2018)
- **MLflow:** provides an intuitive web interface for tracking the results of large experiments and parameter studies. Beyond a few default parameters, a user can include custom parameters to be tracked. (Databricks, 2020)
- **Jupyter Notebook:** the most direct and versatile way to use `Sapsan`.

[2]Please refer to GitHub for the complete list of dependencies.

- **Streamlit (GUI):** a graphical user interface (GUI) for `Sapsan`. While not as flexible as the other interfaces, this can be useful for developing public-facing demonstrations. An example of this interface can be found online at sapsan.app. (Treuille, 2019)
- **Click (CLI):** a command-line interface (CLI) for `Sapsan`. It is used to get the user up and running with templates for a custom project. (Ronacher, 2021)

Applications

While `Sapsan` is designed to be highly customizable for a wide variety of projects in the physical sciences, it is optimized for the study of turbulence. In this section we will demostrate various capabilities of `Sapsan` working with 2D and 3D data, various machine learning libraries, and built-in analytical tools. The ML methods used are included in `Sapsan's` distribution as example `Jupyter notebooks` to get started with the framework.

Hydro simulations

Here is an examples of a turbulence closure model trained on the high-resolution Johns Hopkins Turbulence Database (JHTDB, Li et al., 2008). The training data is a 2D slice of a direct numerical simulation (DNS) of a statistically-stationary isotropic 3D MHD turbulence dataset, 1024^3 in spatial resolution and covering roughly one large eddy turnover time over 1024 checkpoints, i.e. the dynamical time of the system (Eyink et al., 2013). We compare it with a commonly used Dynamic Smagorinsky (DS) turbulence closure model (Lilly, 1966). On the `Sapsan` side, a Kernel Ridge Regression model (Murphy, 2012) by the means of `scikit-learn` is used to demonstrate the effectiveness of conventional ML approaches in tackling turbulence problems. In this test, we used the following setup:

- **Train features:** velocity (u), vector potential (A), magnetic field (B), and their respective derivatives a checkpoint $= 0$. All quantities have been filtered down to 15 Fourier modes to remove small-scale perturbations, mimicking the lower fidelity of a non-DNS simulation. Next they were sampled down to 128^3, with the last step leaving a single slice of 128^2 ready for training.
- **Model Input:** low fidelity velocity (u), vector potential (A), magnetic field (B), and their respective derivatives at a set checkpoint $= 10$.
- **Model Output:** velocity stress tensor component (τ_{xy}) at the matching checkpoint in the future, which effectively represents the difference between large and small scale structures of the system.

In Figure 2, it can be seen that the ML-based approach significantly outperforms the DS subgrid model in reproducing the probability density function, i.e., a statistical distribution of the stress tensor. The results are consistent with (King et al., 2016).

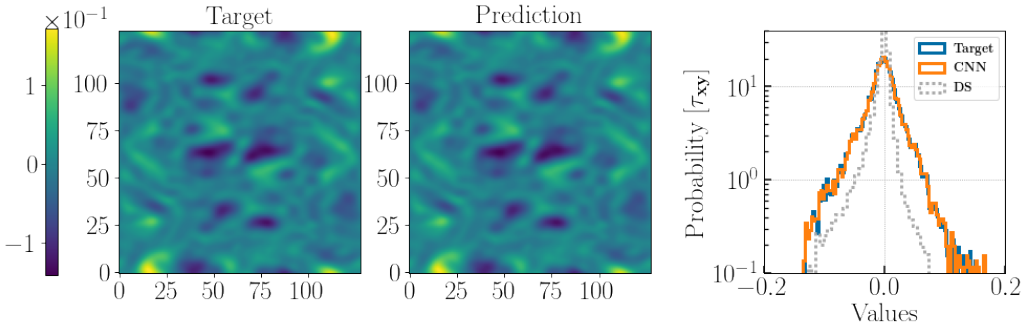

Figure 2: Predicting a 2D turbulent stress-tensor component (τ_{xy}) in statistically-stationary isotropic MHD turbulence setup. The left plot compares the original spatial map of the stress-tensor component to the predicted spatial map (middle). The plot on the right presents probability density functions (PDF), i.e., distributions, of the original stress-tensor component values, the ML predicted values, and the conventional Dynamic Smagorinsky (DS) subgrid model prediction.

Supernovae

Even though the conventional regression-based ML approach worked well in the 2D setup from the previous example, the complexity of our physical problem forced us to seek out a more sophisticated ML method. Supernovae host a different physical regime that is far from the idealistic MHD turbulence case from before. Here we are dealing with dynamically evolving turbulence that is not necessarily isotropic. Turbulence can behave drastically differently depending on the evolutionary stage. With `Sapsan`, we have tested a 3D CNN (Convolutional Neural Network) model built with `PyTorch` to predict a turbulent velocity stress tensor in a realistic Core-Collapse Supernova (CCSN) case. Figure 3 presents results of the following:

- **Train features:** velocity (u), magnetic field (B), and their respective derivatives at time steps before 5 ms (halfway of the total simulation). All quantities have been filtered down with a $\sigma = 9$ Gaussian filter to remove small-scale perturbations, mimicking the lower fidelity of a non-DNS simulation. Lastly they were sampled from the original 348^3 down to 116^3 in resolution.
- **Model Input:** low fidelity velocity (u), magnetic field (B), and their respective derivatives at a set time step in the future beyond 5 ms.
- **Model Output:** velocity stress tensor components (τ_{ij}) at the matching time step in the future, which effectively represents the difference between large and small scale structures of the system.

In this case, the probability density functions are overall consistent, with minor disagreement at the positive outliers, even though the prediction is performed far into the future (time = 9.48 ms, end of the simulation time). Predictive advantage is highlighted when compared with the analytical Gradient model that misses a large portion of positive data.

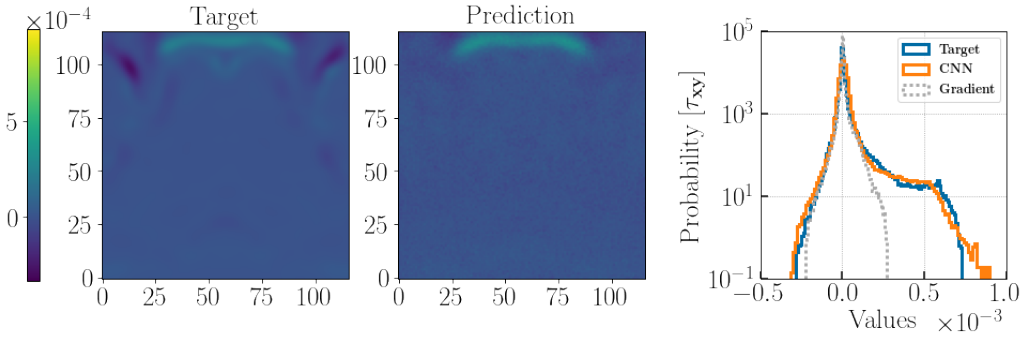

Figure 3: Predicting turbulent stress-tensor component in a core-collapse supernovae (CCSN). The model has been trained on a selection of dynamically evolving turbulence timesteps during the first 5 ms (out of the total ~ 10 ms) of a 3D MHD direct numerical simulation (DNS) after the shockwave bounced off the core in a CCSN scenario. On the left, the two figures are the 2D slices of a 3D τ_{xy} prediction, with the right plot comparing PDFs of the original 3D data, 3D ML prediction, and a conventional Gradient subgrid model.

Acknowledgements

The development of `Sapsan` was supported by the Laboratory Directed Research and Development program and the Center for Space and Earth Science at Los Alamos National Laboratory through the student fellow grant. We would like to thank DOE SciDAC for additional funding support.

References

Carleo, G., Cirac, I., Cranmer, K., Daudet, L., Schuld, M., Tishby, N., Vogt-Maranto, L., & Zdeborová, L. (2019). Machine learning and the physical sciences. *Reviews of Modern Physics*, *91*(4), 045002. https://doi.org/10.1103/RevModPhys.91.045002

Databricks, I. (2020). MLflow. In *GitHub repository*. https://github.com/mlflow/mlflow; GitHub.

Eyink, G., Vishniac, E., Lalescu, C., Aluie, H., Kanov, K., Bürger, K., Burns, R., Meneveau, C., & Szalay, A. (2013). Flux-freezing breakdown in high-conductivity magnetohydrodynamic turbulence. *Nature*, *497*(7450), 466–469. https://doi.org/10.1038/nature12128

Hutson, M. (2018). Artificial intelligence faces reproducibility crisis. *Science*, *359*(6377), 725–726. https://doi.org/10.1126/science.359.6377.725

King, R. N., Hamlington, P. E., & Dahm, W. J. A. (2016). Autonomic closure for turbulence simulations. *Phys. Rev. E*, *93*, 031301. https://doi.org/10.1103/PhysRevE.93.031301

Kolesnikov, S. (2018). Accelerated DL r&d. In *GitHub repository*. https://github.com/catalyst-team/catalyst; GitHub.

Li, Y., Perlman, E., Wan, M., Yang, Y., Meneveau, C., Burns, R., Chen, S., Szalay, A., & Eyink, G. (2008). A public turbulence database cluster and applications to study Lagrangian evolution of velocity increments in turbulence. *Journal of Turbulence*, *9*, N31. https://doi.org/10.1080/14685240802376389

Lilly, D. K. (1966). On the application of the eddy viscosity concept in the Inertial sub-range of turbulence. *NCAR Manuscript 123*.

Liu, S., Meneveau, C., & Katz, J. (1994). On the properties of similarity subgrid-scale models as deduced from measurements in a turbulent jet. *Journal of Fluid Mechanics*, *275*, 83–119. https://doi.org/10.1017/S0022112094002296

Murphy, K. P. (2012). *Machine learning: A probabilistic perspective* (pp. 492–493). The MIT Press.

Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., Killeen, T., Lin, Z., Gimelshein, N., Antiga, L., Desmaison, A., Kopf, A., Yang, E., DeVito, Z., Raison, M., Tejani, A., Chilamkurthy, S., Steiner, B., Fang, L., … Chintala, S. (2019). PyTorch: An imperative style, high-performance deep learning library. In H. Wallach, H. Larochelle, A. Beygelzimer, F. dAlché-Buc, E. Fox, & R. Garnett (Eds.), *Advances in neural information processing systems 32* (pp. 8024–8035). Curran Associates, Inc. http://papers.neurips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-library.pdf

Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., Vanderplas, J., Passos, A., Cournapeau, D., Brucher, M., Perrot, M., & Duchesnay, E. (2011). Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, *12*, 2825–2830.

Ronacher, A. (2021). *Click*. https://click.palletsprojects.com/

Treuille, A. (2019). Turn python scripts into beautiful ML tools. *Towards Data Science*, *8*.

Zhang, W., Zhu, L., Liu, Y., & Kou, J. (2018). Machine learning methods for turbulence modeling in subsonic flows over airfoils. *arXiv e-Prints*, arXiv:1806.05904. http://arxiv.org/abs/1806.05904