A Software Ecosystem for Deploying Deep Learning in Gravitational Wave Physics

Alec Gunny
Dylan Rankin
Philip Harris
Erik Katsavounidis
Ethan Marx
alecg@mit.edu
drankin@mit.edu
pcharris@mit.edu
kats@mit.edu
emarx@mit.edu
Massachusetts Institute of Technology
Cambridge, Massachusetts, USA

Muhammed Saleem
Michael Coughlin
William Benoit
saleem.muhammed.c@gmail.com
cough052@umn.edu
benoi090@umn.edu
School of Physics and Astronomy, University of Minnesota
Minneapolis, Minnesota, USA

ABSTRACT

The recent application of neural network algorithms to problems in gravitational-wave physics invites the study of how best to build production-ready applications on top of them. By viewing neural networks not as standalone models, but as components or functions in larger data processing pipelines, we can apply lessons learned from both traditional software development practices as well as successful deep learning applications from the private sector. This paper highlights challenges presented by straightforward but naïve deployment strategies for deep learning models, and identifies solutions to them gleaned from these sources. It then presents HERMES, a library of tools for implementing these solutions, and describes how HERMES is being used to develop a particular deep learning application which will be deployed during the next data collection run of the International Gravitational-Wave Observatories.

CCS CONCEPTS

• **Applied computing** → **Physics**; • **Computing methodologies** → **Neural networks**.

KEYWORDS

gravitational waves, neural networks, mlops

ACM Reference Format:
Alec Gunny, Dylan Rankin, Philip Harris, Erik Katsavounidis, Ethan Marx, Muhammed Saleem, Michael Coughlin, and William Benoit. 2022. A Software Ecosystem for Deploying Deep Learning in Gravitational Wave Physics. In *Proceedings of the 12th Workshop on AI and Scientific Computing at Scale using Flexible Computing Infrastructures (FlexScience '22), July 1, 2022, Minneapolis, MN, USA.* ACM, New York, NY, USA, 9 pages. https://doi.org/10.1145/3526058.3535454

FlexScience '22, July 1, 2022, Minneapolis, MN, USA
© 2022 Copyright held by the owner/author(s).
ACM ISBN 978-1-4503-9309-6/22/07.
https://doi.org/10.1145/3526058.3535454

1 INTRODUCTION

The last several years have witnessed an explosion in the application of deep learning algorithms to problems in the physical sciences [17–19, 27, 30]. Gravitational-wave physics has proven to be especially fertile ground for this research [19, 26], where efficient, low-latency detection and parameterization of gravitational-wave events [4, 9] has the potential to enable a host of discoveries during the next data collection run of the observatories of the International Gravitational-Wave Observatory Network (IGWN), referred to as Observing Run 4 (O4) [2, 5, 25]. However, much of this work has focused primarily on showing as a proof-of-concept *that* deep learning could supplant various existing modelling techniques in gravitational-wave analysis, with less attention to more practical questions about *how* this ecosystem of models, deep learning and otherwise, might work together to power real discoveries during O4.

This research has emerged in a decade in which deep learning has been shown more broadly to achieve state-of-the-art results in fields such as computer vision [36, 37], natural language processing [22, 31], generative modelling [33], and complex game playing [13, 28]. However, for much of this time, private sector applications of deep learning suffered the same curse of "AI for AI's sake" [6], with only a handful of the largest companies deploying production deep learning applications at scale. However, recent years have witnessed a growth of mature deep learning applications powered by the emerging field of machine learning operations (MLOps) [7]. This field seeks to develop a set of tools and best practices which reduce the overhead associated with responsibly training, testing, and deploying deep learning applications [32]. These practices in turn enable domain experts in novel fields to replace hand-crafted functions with off-the-shelf neural networks in new or existing applications, taking advantage of improved performance, real-time inference capabilities, or simplified engineering with a minimum of friction. While there remains much work to be done to establish if and how deep learning algorithms will fit into the data processing pipelines of large-scale physics experiments alongside or in place of more traditional models, their clear success on other problems and promising results in this space thus far suggest that studying these practices in this context is worthwhile.

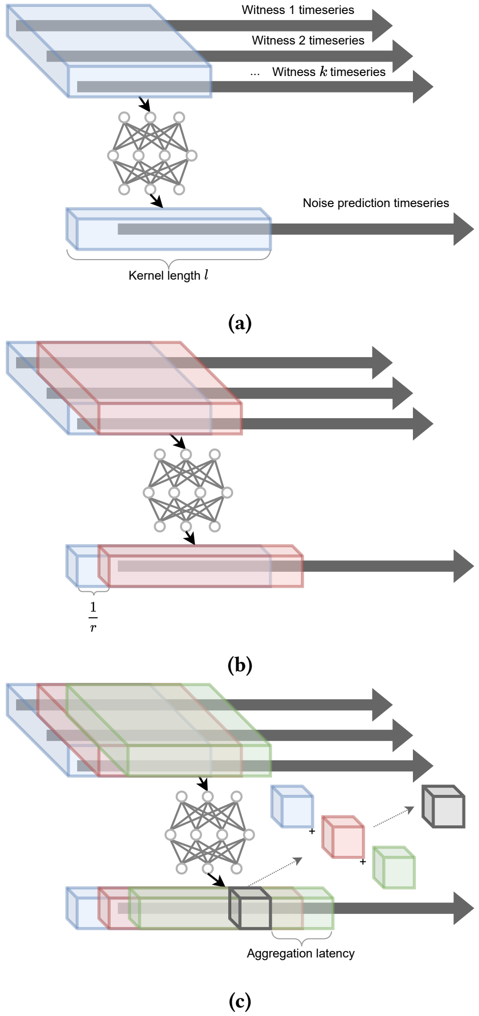
In this work, we describe the key challenges we faced in developing a deep learning-based application in the context of the IGWN. We then identify how emerging software and infrastructure tools can address these challenges, highlighting available off-the-shelf components when applicable. We also outline how the unique constraints of the gravitational-wave data analysis problem and computing environment demand more custom solutions, and introduce HERMES, a library of open-sourced tools designed to make these solutions available to practitioners more generally. Finally, we conclude by describing a deep learning application built on top of these tools which will be in deployment in O4, taking advantage of modern MLOps practices to more quickly take novel ideas into production with higher degrees of confidence in their expected behavior.

2 BACKGROUND

We begin by briefly providing background on the IGWN data processing setting more generally, as well as on the specific problem within it to which we have applied deep learning. LIGO [2, 23] consists of two ground-based detectors located in the United States whose purpose is to detect minuscule fluctuations in the fabric of spacetime called gravitational waves. In conjuction with the Kamioka Gravitational Wave Detector (KAGRA) in Japan [8, 35] and the Virgo interferometer in Italy[5], these observatories will begin a joint fourth run of observation, the O4 run, in early 2023 with the hope of detecting and characterizing gravitational wave events with unparalleled sensitivity and precision [3]. The ability to perform this detection and characterization at low-latency will be critical to the use of these observatories as early-warning triggers for follow-up by observatories of other cosmic messengers such as electromagnetic radiation, cosmic rays, and neutrinos. Comparison of measurements across these disparate modalities from the same events, referred to as *multi-messenger astrophysics* (MMA), will help deliver a slew of new insights during O4[14].

The detectors in these observatories use very finely-calibrated lasers to measure timeseries of a unitless quantity called gravitational-wave strain, denoted $h(t)$. However, the delicate calibration on which the detector's sensitivity relies is constantly disturbed by factors from the environment surrounding its instruments [10, 38]. In order to mitigate the influence of these environmental conditions, sensors are deployed to monitor them so that the timeseries measured by these *witness channels* can be used to predict the corresponding noise observed in the strain channel. DeepClean [29] attempts to model this mapping from witnesses to noise using a neural network in order to capture non-linear couplings between the channels in real-time in order to meet the demands of MMA.

The architecture underlying DeepClean follows a convolutional autoencoder-like structure, mapping finite-length snapshots or *kernels* of the k witness timeseries to a kernel of predicted witnessed noise of the same length. Because DeepClean encodes entirely local information, the size of the kernel l is not fixed and can be chosen as an inference-time parameter to trade off between compute and prediction quality. Moreover, the rate at which these kernels are sampled from the timeseries is another inference-time parameter, which we refer to as the *inference sampling rate* r (not to be confused with the rate f_s at which the timeseries itself is sampled). If the inference sampling rate is high compared to the length of the kernel, $\frac{1}{r} < l$, there will be overlap between the input and output kernels as depicted in figure 1.



(a)

(b)

(c)

Figure 1: Performing inference with DeepClean. (a) DeepClean maps from finite-length snapshots or kernels of witness timeseries to a kernel of the same length representing a prediction of the witnessed noise in the strain channel at that same snapshot. (b) Kernels are sampled from the timeseries at some fixed frequency $r \leq f_s$, the sample rate of the data. If $\frac{1}{r} < l$, there will be overlap between the input and output kernels. (c) Overlapping predictions are averaged over a fixed number of kernels in order to improve prediction quality, introducing latency as future data is generated with which to average.

3 MOTIVATION

While the original DeepClean inference implementation which produced the results outlined in [29] proved itself capable of adding value to the noise subtraction problem, it also made several modelling and software decisions which do not lend themselves to the online deployment scenario required by a real-time production application. In this section, we outline some of these difficulties and show how they lead to a new set of infrastructure tools as solutions.

3.1 Inference-as-a-Service

A production DeepClean application is fundamentally a data processing pipeline charged with removing environmental noise from a streaming timeseries of strain data. As one of its components, this application makes use of an inference function which it uses to map from witness noise timeseries to an estimated noise timeseries in a "black-box" fashion. The rest of the application encodes logic about how to load and prepare data to feed this inference function, and how to turn its outputs into a physically significant timeseries which can be subtracted from the raw strain channel. That the inference function leverages a neural network, rather than a more explicitly modelled function, is a detail to which the physics encoded in the cleaning application ought to be agnostic.

The most straightforward way to take a trained DeepClean network and use it in an application is to load it into the application's memory and use it to perform inference locally. However, this intuitive approach has several drawbacks, outlined in figure 2a. Coordinating the distribution of the piece of software which a deployed neural network represents leads to inconsistent results as users become out of sync with one another and the latest changes to the model. Moreover, the deep learning software stack is complex and places a non-trivial burden on users to gain familiarity with the requisite libraries and high-performance compute techniques in order to effectively leverage expensive heterogeneous computing resources, to which they may not even have easy access. All of these issues are made combinatorially more difficult in even slightly more complex scenarios that involve utilizing multiple neural networks which leverage different software frameworks.

Perhaps more importantly in the physical sciences, gaining familiarity with deep learning techniques and software involves a significant investment of time in skills which may be orthogonal to a practitioner's field of interest. While some physicists may find gaining this expertise valuable, requiring it as a price of admission for *any* user will deter adoption of important algorithms and ultimately hamper discoveries. Good deep learning inference implementations ought to extricate these details from the applications that leverage them in order to allow the physicists building them to focus on their domains of expertise.

A popular emerging paradigm to address these difficulties is the inference-as-a-service (IaaS) model, in which inference is performed by an off-the-shelf application which can efficiently schedule asynchronous, framework-agnostic inference executions across heterogeneous hardware platforms [15]. An IaaS deployment is illustrated in figure 2b, with a centralized model repository keeping all users in-sync and up-to-date. Instead of making local inference calls, pipelines send gRPC inference requests to the service using standardized APIs which abstract the implementation details of the inference execution itself. One such IaaS application is NVIDIA's Triton Inference Server[1], which is well optimized for GPU performance and supports multiple framework backends as well as model ensembling. By deploying Triton via software containers, we are able to wrap up the inference portion of our cleaning application as a self-contained, portable black-box which is usable by both our application and other users.

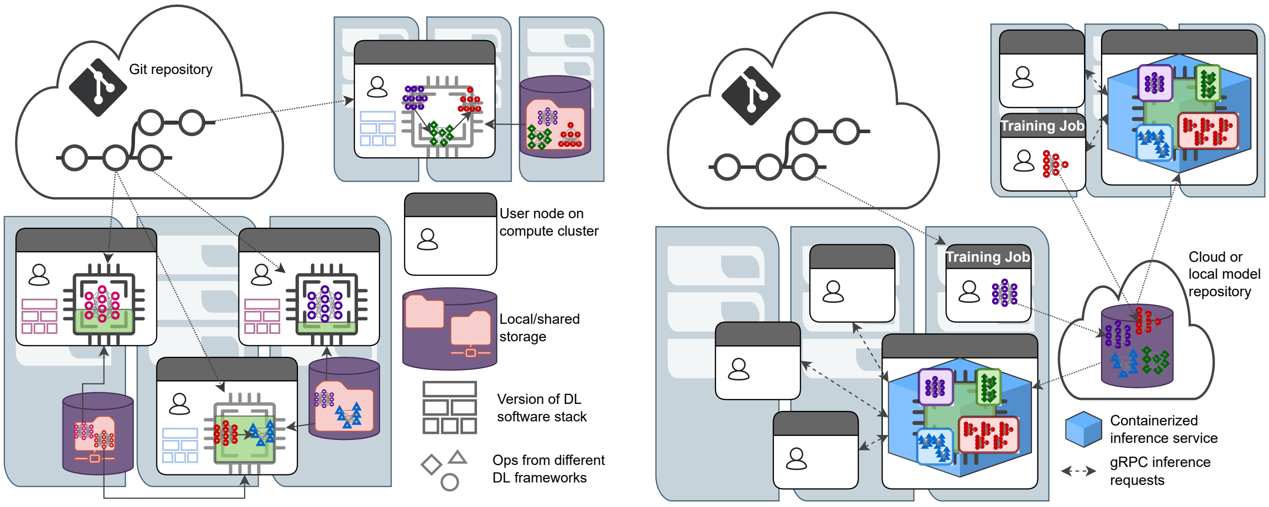
While Triton helped to massively simplify the deployment of a trained DeepClean model, it introduced new issues as well. Triton expects the model repository from which it loads the models it exposes for inference to follow a strict structure, which involves a fair amount of systematic hand coding (boilerplate) to ensure that new entries and versions are added to the repository in the appropriate fashion. Moreover, it relies on configuration files implemented as protocol buffers, whose programming involves non-Pythonic syntax and requires information that can often be dynamically inferred from the model itself, resulting in more boilerplate. Triton's client APIs also suffer from these boilerplate issues, especially when implementing the streaming inference required by DeepClean's caching model outlined in section 5.3. Components of the `hermes` library discussed in section 4 were built to address these issues in order to more smoothly export and make requests to new or retrained models.

3.2 Online deployment

While an IaaS deployment solved the problems in our pipeline surrounding deep learning inference, other components still needed to be updated to reflect the online inference scenario. In the original offline inference scenario, each segment in the timeseries of DeepClean's noise predictions is produced by averaging over predictions made on that segment by *all* kernels, including those which contain data from the future. Moreover, it performs forward and bandpass filtering over the entire timeseries of noise predictions before subtracting it from the strain timeseries. Online inference, however, has a much higher sensitivity to additional latency incurred waiting for future data to materialize, and as such requires different modelling choices. At first, our instinct was to neglect averaging altogether and filter using only past data in order to minimize latency. However, these choices produced vastly different results than those made in the offline scenario, results which introduced more noise than they removed.

In contrast, we can average and bandpass filter over some, but not all, future data to gain higher quality predictions in exchange for some latency. This amount of data, along with the optimal kernel length l and inference sampling rate r, amount to parameters of a broader *M*odel being optimized, of which the DeepClean neural network *m*odel is but a component. A more complex, truly production-ready application that involves network retraining will also have to optimize parameters deciding how frequently to retrain, how to select or search over retraining hyperparameters, which layers of the network to retrain, and which metrics and thresholds to use to decide whether a retrained network is ready for deployment. It will also need to make decisions about how to handle dropped witness channels, shifting data distributions, and other as-yet-unforeseen eventualities which might impact the quality of its cleaned output.

[1] https://github.com/triton-inference-server/server


Git repository
User node on compute cluster
Local/shared storage
Version of DL software stack
Ops from different DL frameworks
Training Job
Training Job
Cloud or local model repository
Containerized inference service
gRPC inference requests

(a) Traditional distributed deployment scenario

(b) Inference-as-a-Service deployment scenario

Figure 2: (a) A traditional deployment scenario in which individual users manage their own software and hardware resources. Inconsistencies in libraries and dependencies as well as model versions lead to inconsistent results. Reduced computational demands of inference lead to hardware under-utilization, represented by green rectangles on each node. More complex deployment scenarios require leveraging multiple networks utilizing multiple framework backends, exacerbating existing issues. (b) Inference-as-a-service deployment standardizes inference across all users and coordinates complex concurrent execution of models, saturating hardware compute capacity in a way that is portable and scalable.

Good MLOps infrastructure reduces the overhead required to deploy novel ideas in controlled environments that *look like* the true test environment, and evaluate them using metrics that *look like* the true test metrics. This allows us to optimize the Model as a whole, rather than one of its components at some fixed point in time, and formalize contractual obligations to downstream users by creating confident estimates of performance on the true test distribution [12, 24]. Moreover, adopting good MLOps practices like versioning experiments and automating their deployment via continuous integration (CI) ensures that results are repeatable, comparable, and conclusive [32].

4 HERMES

In order to simplify the development of applications that implement the ideas outlined in Section 3, we have developed a set of Python libraries collectively called `hermes` [20]. These libraries provide simple, intuitive interfaces for executing many of the tasks surrounding deep learning deployment, including model export and acceleration, asynchronous data processing and inference request generation, and cloud-based resource provisioning and deployment. Taken together, the `hermes` libraries are intended to provide the building blocks on which higher-level, problem-specific abstractions can be built to further streamline the process of deploying deep learning-based applications. Section 5 outlines how our production DeepClean deployment realizes this potential.

4.1 The HERMES libraries

HERMES consists of multiple sub-libraries, each with their own dedicated functionality and corresponding dependencies, allowing users to pick and choose which libraries they need in order to keep their deployments lightweight. In this section, we will briefly describe the purpose of the most relevant libraries, and leave more detailed information to their documentation.

4.1.1 `hermes.cloudbreak`. The `cloudbreak` library contains tools for provisioning Kubernetes clusters and virtual machines on private clouds and deploying workloads onto those computational resources. While support only currently exists for Google Cloud, the intent of the library is to be written in such a way that the user interface is agnostic to the actual cloud backend. Moreover, by using Python contexts to deploy resources, we can ensure that any resources are spun-down once jobs are complete so that unnecessary costs are not incurred. `cloudbreak` is not currently used as part of the DeepClean production pipeline, but will form a critical part of future large-scale offline experiments.

4.1.2 `hermes.quiver`. The `quiver` library is used to reduce the user-written boilerplate associated with exporting a model for use with Triton outlined in section 3.1. It takes care of structuring and exporting models to your model repositories, either locally or in the cloud, and uses the in-memory version of your model to extract all the necessary metadata to build the associated configuration file Triton expects. It can also automatically facilitate the conversion of models from common frameworks like Torch to the accelerated inference library TensorRT[2], and has utilities for quickly constructing model ensembles and exposing input and output server-side caching models for minimizing data transfer in streaming use cases. This latter functionality allows us to perform

[2] https://developer.nvidia.com/tensorrt

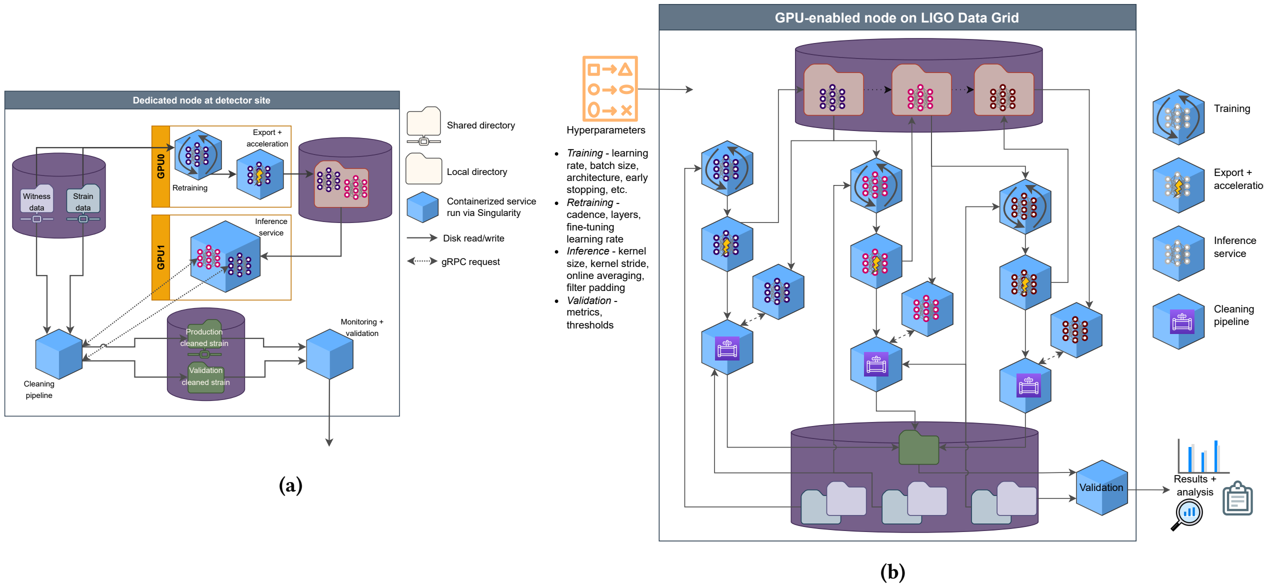

Dedicated node at detector site
GPU0
Export + acceleration
Retraining
GPU1
Inference service
Witness data
Strain data
Production cleaned strain
Validation cleaned strain
Monitoring + validation
Cleaning pipeline
Shared directory
Local directory
Containerized service run via Singularity
Disk read/write
gRPC request
(a)
GPU-enabled node on LIGO Data Grid
Hyperparameters
Training - learning rate, batch size, architecture, early stopping, etc.
Retraining - cadence, layers, fine-tuning learning rate
Inference - kernel size, kernel stride, online averaging, filter padding
Validation - metrics, thresholds
Training
Export + acceleration
Inference service
Cleaning pipeline
Validation
Results + analysis
(b)

Figure 3: (a) A production DeepClean deployment. The inference service hosts production and development versions of the model, with the development model being moved into production once its performance has been validated on the live data stream. New models are produced at a fixed cadence or in response to instrument interruptions in order to keep the model up-to-date with nonstationary data distributions. Cleaned data are written to shared directories to be made available to downstream users via automated data distribution mechanisms. (b) DeepClean offline experimentation structure. Each component of the production application is run in serial in containers hosting the experimental version of the code. The test data used to validate one version of the model is used to train the next version. The cleaned strain produced at each model testing is consumed by a validation application which produces a publishable document containing experimental results and analyses for discussion in the corresponding pull request.

online averaging of DeepClean's outputs on the server-side, streaming back non-overlapping averaged predictions for each segment and avoiding the drawbacks of the "fully online" inference scenario depicted in figure 14 of [21]. More information on this will be given in section 5.3.

4.1.3 `hermes.stillwater`*.* One of the benefits of leveraging a dedicated inference service is that it allows the deep learning inference step of a pipeline to execute asynchronously from the other steps, increasing the total throughput capacity. `stillwater` helps client pipelines take advantage of this by offering a `PipelineProcess` class which simplifies implementing data loading and pre- and post-processing steps as asynchronous processes which communicate via pipes. It also provides a `InferenceClient` subclass which addresses some more of the boilerplate referenced in section 3.1 by dynamically inferring the names, shapes, and datatypes of the inputs expected by a given network for making asynchronous streaming requests.

5 DEEPCLEAN IN PRODUCTION

5.1 Deployment Outline

Figure 3a illustrates the functionality of a fully production-ready DeepClean application. Application components are deployed with containers via Singularity [11] and orchestrated via Singularity Compose [34]. Because of the real-time demands of this application, all components are deployed on the same node at each detector site in order to minimize communication latency. Raw witness and strain data are made available in 1-second increments in a shared drive through automated gravitational-wave data distribution mechanisms. Cleaned strain data is written to similar shared drives in order to leverage the same distribution mechanisms for downstream users.

After the network is fine-tuned on new data in order to keep up with nonstationary data distributions, it is accelerated via TensorRT, and deployed as a development version on the inference service via `hermes.quiver`. A secondary data processing stream in the cleaning pipeline, implemented with `hermes.stillwater`, writes data cleaned using this development model to a local directory. A monitoring service measures metrics of interest on both the production and development data, ensuring that the production deployment is maintaining adequate performance and publishing its metrics to an internal site at a fixed cadence. It also validates the performance of the development model and ensures that it can meet established standards. Once this is validated, the current production model is retired to cold storage and the development model is moved into production.

5.2 Offline development

Novel additions to the DeepClean analysis pipeline will require thorough validation in an environment which replicates the full

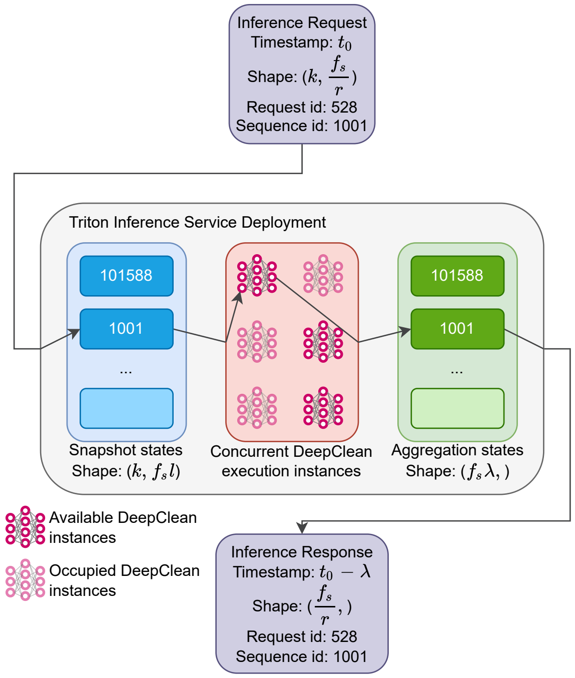


Figure 4: Triton deployment ensemble. A snapshotter model maintains the state of each active timeseries' most recent input kernel, while an aggregator model maintains the associated online average of DeepClean predictions on overlapping segments. Inference requests contain only updates to the existing state, and are associated with a unique sequence identifier which Triton uses to perform the correct update. Multiple DeepClean instances are hosted on the inference service to perform concurrent inference on different sequences and kernels from the same sequence simultaneously, helping to saturate GPU utilization. States for both the snapshotter and aggregator models are initialized to 0 for new sequences. If no states are available, new sequences are queued until an existing occupied state times out or sends a "sequence end" flag with its request. Aggregated outputs have the same length as input requests, but with a time delay between their initial timestamp created by the aggregation process.

test-time deployment scenario and which computes those metrics which represent DeepClean's contractual obligations to downstream users. Experiments which test these hypothesized additions are deployed automatically to the International Gravitational Wave Network Computing Grid (IGWN Grid) via the methods outlined in section 5.5, which allows these experiments to be versioned alongside the code which implements them. Figure 3b illustrates what such an experiment might look like, executing the asynchronous steps performed by the various DeepClean application components in serial to aid debugging and analysis, and automating the measurement of performance metrics. Many such pipelines may be run during a single experiment to optimize over the set of relevant hyperparameters. In this case, the pipeline should be run again on a held-out segment of data using the optimized hyperparameters to validate algorithm performance.

5.3 Streaming inference

As described in [21], the streaming nature of gravitational-wave data complicates the use of inference services. Overlapping data in both the input and output kernels introduces extra data transfer overhead which is linear in the inference sampling rate r. At sufficiently high values of r, this overhead becomes the dominant source of inference latency.

Figure 4 illustrates how this issue is resolved in the production inference server deployment. The `hermes.quiver` library is used to construct a Triton ensemble with a "snapshotter" model up front which maintains the most recent kernel for a given timeseries as a state on the server associated with a specific sequence identifier. Inference requests are sent containing only *new* data which is used to update this state. Triton routes the update to the appropriate state using a sequence identifier attached to the request. This updated kernel is passed to a pool of DeepClean inference instances, which Triton manages to perform concurrent inference so that subsequent kernels can be inferred upon simultaneously. This is critical to our use case because the 1-second increments in which data is made available means that we can stream updates to the inference service faster than inference can be performed on them.

The one extension we make here beyond the work outlined in [21] is that rather than adopting a fully online inference scheme to address the data transfer bottleneck, we implement another stateful model in the ensemble which maintains an online average of DeepClean's predictions on overlapping segments. This allows us to take advantage of the higher quality cleans associated with averaged predictions, without transferring redundant data. This *aggregator* model streams back update-sized segments of data once they have averaged over a fixed number of DeepClean predictions. This necessarily introduces some aggregation latency λ between the initial timestamp of the inference *request* which triggered the inference, and that of the inference *response* which gets streamed back, since the segment of data with the request's initial timestamp has not averaged over enough predictions yet.

5.4 Environment management

Managing the software environments in which these experiments are executed is nontrivial. We use Poetry[3] to manage most dependencies due to its ease of use for projects with the *monorepo* structure that DeepClean has adopted. For projects like this, Poetry makes it easy to manage local libraries with editable installs to accelerate the development process. However, there are packages for reading and writing files specific to gravitational-wave analysis that are only installable via Anaconda [1]. Keeping track of how and when to leverage these different environment management tools is onerous and decreases the repeatability of environment builds as different methods of reconciling them are employed.

To address this, we have built a command line utility `pinto`[4] which alleviates these difficulties by automatically using a project's configuration to decide how to create and manage its corresponding

[3] https://github.com/python-poetry/poetry
[4] https://github.com/ML4GW/pinto

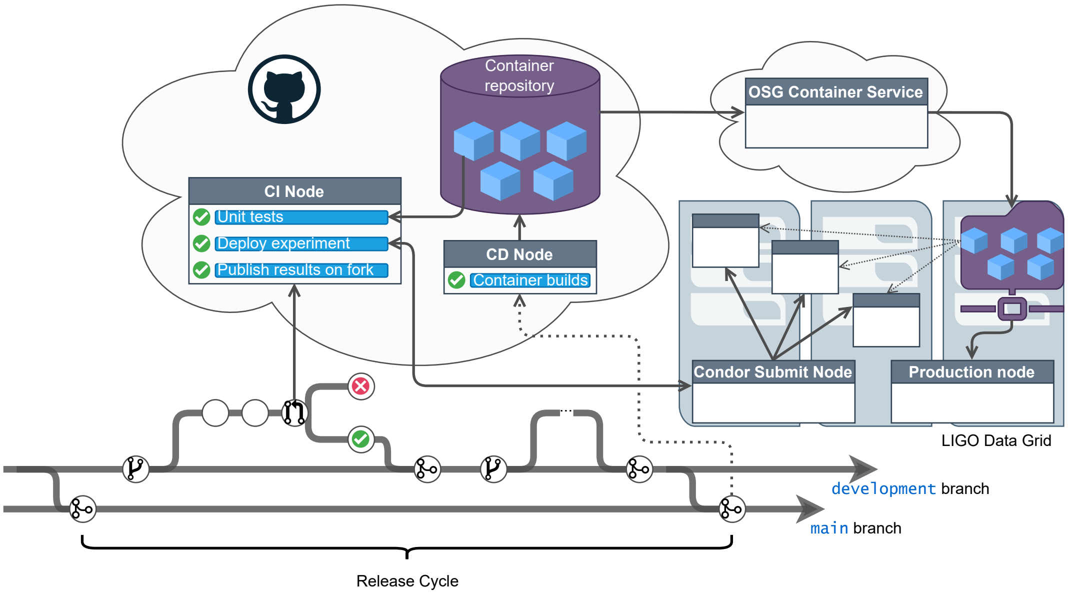


Figure 5: DeepClean development cycle. Pull requests automatically deploy experiments on the LIGO Data Grid so that they are versioned along with the code that implements them. Analysis of novel results is deployed to a public web page for discussion and comment. Accepted changes are reflected in the deployment containers on a fixed release cadence. A container deployment service from the Open Science Grid publishes latest releases to production nodes.

software environment and execute its commands inside of it. Pinto's development cycle is managed separately from both HERMES and DeepClean to be able to service both training and deployment for other deep learning-based gravitational-wave analysis projects. It is built and packaged as a container by leveraging GitHub's continuous deployment (CD) tools, and is used for unit testing DeepClean components. This container image is also used as the base image on top of which several of DeepClean's production containers are built, each of which is pinned to a specific version to ensure the asynchronous development cycles don't disrupt DeepClean's functionality.

5.5 Development Cycle

Figure 5 depicts the development cycle of the production DeepClean application. Bug fixes, feature additions, and new ML research take place in branches forked from an upstream `development` branch. Once a forked branch opens a pull request to the `development` branch, GitHub's CI tools use the Pinto container to test the code then submit the relevant experiment as a job or set of jobs (e.g. for hyperparameter searching) on the IGWN Grid. A local version of the Pinto container which matches the version used for testing executes the steps of the experiment on the IGWN Grid. The experiment's results and analysis are passed back as an HTML file to the CI job which publishes them to the GitHub Page[5] of the forked repository to facilitate discussion on the merits of the changes. Once there is consensus among the repository maintainers as to whether to accept or reject the proposed changes, the pull request is merged or closed, respectively. A merged pull request will then trigger the same CI steps for DeepClean's standard experiments, which may have been changed or augmented in the pull request. The results of the experiment are published to the upstream repository's documentation hosted on its GitHub Page so that the latest information is always available to users.

The `development` branch is merged into the `main` branch at a fixed cadence, with each new merge triggering a new release of the production containers. These containers are also published to GitHub's container repository, then made available on our dedicated node on the IGWN Grid via Open Science Grid's[16] container sync service[6]. The two latest versions of each container are always made available on the IGWN Grid, in case unexpected bugs force us to roll back to a previous stable version. Once new containers become available, the service is restarted to leverage the new software.

6 CONCLUSION

We have outlined in this work the requirements of a production-ready, real-time deep learning application in the context of gravitational wave analysis. We have outlined how existing off-the-shelf tools for constructing such an application are insufficient to meet the demands of a robust development ecosystem, and have described an open-sourced set of tools we have built to address these difficulties. Finally, we have described how these tools are being

[5] https://pages.github.com/

[6] https://github.com/opensciencegrid/cvmfs-singularity-sync

used to deploy a mature application for performing noise subtraction during the O4 data collection run of the IGWN observatories.

While several of the components of this application are still under development, we hope that by releasing the application and relevant tools in the open-source we can both facilitate similar work by our peers and leverage their contributions to realize a more robust system. In the longer term, our secondary goal is to deploy a cloud-based DeepClean inference application that can be requested by arbitrary authorized users in order to integrate more tightly into downstream detection and parameter estimation pipelines. By building the components of this simpler application correctly, our hope is that such a goal can be achieved by straightforward extension of the same tools, bringing the power of modern deep learning algorithms to a wider base of users with diverse expertise.

ACKNOWLEDGMENTS

All authors acknowledge support from the National Science Foundation with grant numbers OAC-1931469, OAC-1934700, PHY-2010970 and OAC-2117997. W.B. additionally acknowledges support through DGE-1922512. This material is based upon work supported by NSF's LIGO Laboratory which is a major facility funded by the National Science Foundation. The authors are grateful for computational resources provided by the LIGO Laboratory and supported by National Science Foundation Grants PHY-0757058 and PHY-0823459.

REFERENCES

[1] 2020. Anaconda Software Distribution. https://docs.anaconda.com/

[2] J. Aasi et al. 2015. Advanced LIGO. *Classical and Quantum Gravity* 32, 7 (2015), 074001.

[3] B. P. Abbott et al. 2018. Prospects for observing and localizing gravitational-wave transients with Advanced LIGO, Advanced Virgo and KAGRA. *Living Rev. Rel.* 21, 1 (2018), 3. https://doi.org/10.1007/s41114-020-00026-9 arXiv:1304.0670 [gr-qc]

[4] B. P. Abbott et al. 2019. Low-latency Gravitational-wave Alerts for Multimessenger Astronomy during the Second Advanced LIGO and Virgo Observing Run. *Astrophys. J.* 875, 2 (2019), 161. https://doi.org/10.3847/1538-4357/ab0e8f arXiv:1901.03310 [astro-ph.HE]

[5] F. Acernese et al. 2015. Advanced Virgo. *Classical and Quantum Gravity* 32, 2 (2015), 024001.

[6] Algorithmia Inc. 2019. Algorithmia 2020 state of enterprise machine learning. Alogrithmia Inc., Seattle, WA, USA. https://info.algorithmia.com/hubfs/2019/Whitepapers/The-State-of-Enterprise-ML-2020/Algorithmia_2020_State_of_Enterprise_ML.pdf

[7] Algorithmia Inc. 2020. Algorithmia 2021 state of enterprise machine learning. Alogrithmia Inc., Seattle, WA, USA. https://info.algorithmia.com/hubfs/2020/Reports/2021-Trends-in-ML/Algorithmia_2021_enterprise_ML_trends.pdf

[8] Yoichi Aso, Yuta Michimura, Kentaro Somiya, Masaki Ando, Osamu Miyakawa, Takanori Sekiguchi, Daisuke Tatsumi, and Hiroaki Yamamoto. 2013. Interferometer design of the KAGRA gravitational wave detector. *Phys. Rev. D* 88 (Aug 2013), 043007. Issue 4. https://doi.org/10.1103/PhysRevD.88.043007

[9] Kipp Cannon et al. 2012. Toward Early-Warning Detection of Gravitational Waves from Compact Binary Coalescence. *Astrophys. J.* 748 (2012), 136. https://doi.org/10.1088/0004-637X/748/2/136 arXiv:1107.2665 [astro-ph.IM]

[10] Davis et al. 2021. LIGO detector characterization in the second and third observing runs. *Classical and Quantum Gravity* 38, 13 (jun 2021), 135014. https://doi.org/10.1088/1361-6382/abfd85

[11] SingularityCE Developers. 2021. *SingularityCE 3.8.3.* https://doi.org/10.5281/zenodo.5564915

[12] Alexander D'Amour et al. 2020. Underspecification Presents Challenges for Credibility in Modern Machine Learning. https://doi.org/10.48550/ARXIV.2011.03395

[13] David Silver et al. 2017. Mastering the game of Go without human knowledge. *Nature* 550, 7676 (Oct. 2017), 354–359. https://doi.org/10.1038/nature24270

[14] E. A. Huerta et al. 2019. Enabling real-time multi-messenger astrophysics discoveries with deep learning. *Nature Reviews Physics* 1, 10 (oct 2019), 600–608. https://doi.org/10.1038/s42254-019-0097-4

[15] Javier Duarte et al. 2019. FPGA-Accelerated Machine Learning Inference as a Service for Particle Physics Computing. *Computing and Software for Big Science* 3, 1 (oct 2019). https://doi.org/10.1007/s41781-019-0027-2

[16] Ruth Pordes et al. 2007. The open science grid. *Journal of Physics: Conference Series* 78 (jul 2007), 012057. https://doi.org/10.1088/1742-6596/78/1/012057

[17] Matthew Feickert and Benjamin Nachman. 2021. A Living Review of Machine Learning for Particle Physics. https://doi.org/10.48550/ARXIV.2102.02770

[18] Christopher J. Fluke and Colin Jacobs. 2019. Surveying the reach and maturity of machine learning and artificial intelligence in astronomy. *WIREs Data Mining and Knowledge Discovery* 10, 2 (dec 2019). https://doi.org/10.1002/widm.1349

[19] Daniel George and E.A. Huerta. 2018. Deep Learning for real-time gravitational wave detection and parameter estimation: Results with Advanced LIGO data. *Physics Letters B* 778 (mar 2018), 64–70. https://doi.org/10.1016/j.physletb.2017.12.053

[20] Alec Gunny and Dylan Rankin. 2021. *fastmachinelearning/gw-iaas: v0.1.0-alpha.* https://doi.org/10.5281/zenodo.5567703

[21] Alec Gunny, Dylan Rankin, Jeffrey Krupa, Muhammed Saleem, Tri Nguyen, Michael Coughlin, Philip Harris, Erik Katsavounidis, Steven Timm, and Burt Holzman. 2021. Hardware-accelerated Inference for Real-Time Gravitational-Wave Astronomy. https://doi.org/10.48550/ARXIV.2108.12430

[22] Jesse Michael Han, Igor Babuschkin, Harrison Edwards, Arvind Neelakantan, Tao Xu, Stanislas Polu, Alex Ray, Pranav Shyam, Aditya Ramesh, Alec Radford, and Ilya Sutskever. 2021. Unsupervised Neural Machine Translation with Generative Language Models Only. https://doi.org/10.48550/ARXIV.2110.05448

[23] Gregory M Harry et al. 2010. Advanced LIGO: the next generation of gravitational wave detectors. *Classical and Quantum Gravity* 27, 8 (apr 2010), 084006. https://doi.org/10.1088/0264-9381/27/8/084006

[24] Alon Jacovi, Ana Marasović, Tim Miller, and Yoav Goldberg. 2020. Formalizing Trust in Artificial Intelligence: Prerequisites, Causes and Goals of Human Trust in AI. https://doi.org/10.48550/ARXIV.2010.07487

[25] KAGRA collaboration. 2019. KAGRA: 2.5 generation interferometric gravitational wave detector. *Nature Astronomy* 3, 1 (2019), 35–40. https://doi.org/10.1038/s41550-018-0658-y

[26] Plamen G. Krastev, Kiranjyot Gill, V. Ashley Villar, and Edo Berger. 2021. Detection and parameter estimation of gravitational waves from binary neutron-star mergers in real LIGO data using deep learning. *Physics Letters B* 815 (2021), 136161. https://doi.org/10.1016/j.physletb.2021.136161

[27] M. Ntampaka, J. ZuHone, D. Eisenstein, D. Nagai, A. Vikhlinin, L. Hernquist, F. Marinacci, D. Nelson, R. Pakmor, A. Pillepich, P. Torrey, and M. Vogelsberger. 2019. A Deep Learning Approach to Galaxy Cluster X-Ray Masses. *The Astrophysical Journal* 876, 1 (may 2019), 82. https://doi.org/10.3847/1538-4357/ab14eb

[28] OpenAI, :, Christopher Berner, Greg Brockman, Brooke Chan, Vicki Cheung, Przemysław Dębiak, Christy Dennison, David Farhi, Quirin Fischer, Shariq Hashme, Chris Hesse, Rafal Józefowicz, Scott Gray, Catherine Olsson, Jakub Pachocki, Michael Petrov, Henrique P. d. O. Pinto, Jonathan Raiman, Tim Salimans, Jeremy Schlatter, Jonas Schneider, Szymon Sidor, Ilya Sutskever, Jie Tang, Filip Wolski, and Susan Zhang. 2019. Dota 2 with Large Scale Deep Reinforcement Learning. https://doi.org/10.48550/ARXIV.1912.06680

[29] Rich Ormiston, Tri Nguyen, Michael Coughlin, Rana X. Adhikari, and Erik Katsavounidis. 2020. Noise reduction in gravitational-wave data via deep learning. *Physical Review Research* 2, 3 (jul 2020). https://doi.org/10.1103/physrevresearch.2.033066

[30] Fernanda Psihas, Micah Groh, Christopher Tunnell, and Karl Warburton. 2020. A review on machine learning for neutrino experiments. *International Journal of Modern Physics A* 35, 33 (nov 2020), 2043005. https://doi.org/10.1142/s0217751x20430058

[31] Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, and Peter J. Liu. 2019. Exploring the Limits of Transfer Learning with a Unified Text-to-Text Transformer. https://doi.org/10.48550/ARXIV.1910.10683

[32] Khalid Salama, Jarek Kazmierczak, and Donna Schut. May 2021. Practitioners guide to MLOps: A framework for continuous delivery and automation of machine learning. Google Inc., Mountain View, CA, USA. https://services.google.com/fh/files/misc/practitioners_guide_to_mlops_whitepaper.pdf

[33] Axel Sauer, Katja Schwarz, and Andreas Geiger. 2022. StyleGAN-XL: Scaling StyleGAN to Large Diverse Datasets. https://doi.org/10.48550/ARXIV.2202.00273

[34] Vanessa Sochat. 2019. *Singularity Compose: Orchestration for Singularity Instances.* https://doi.org/10.5281/zenodo.3376793

[35] Kentaro Somiya. 2012. Detector configuration of KAGRA–the Japanese cryogenic gravitational-wave detector. *Classical and Quantum Gravity* 29, 12 (2012), 124007.

[36] Chien-Yao Wang, I-Hau Yeh, and Hong-Yuan Mark Liao. 2021. You Only Learn One Representation: Unified Network for Multiple Tasks. https://doi.org/10.48550/ARXIV.2105.04206

[37] Yuhui Yuan, Xiaokang Chen, Xilin Chen, and Jingdong Wang. 2019. Segmentation Transformer: Object-Contextual Representations for Semantic Segmentation. (2019). https://doi.org/10.48550/ARXIV.1909.11065

[38] M Zevin, S Coughlin, S Bahaadini, E Besler, N Rohani, S Allen, M Cabero, K Crowston, A K Katsaggelos, S L Larson, and et al. 2017. Gravity Spy: integrating advanced LIGO detector characterization, machine learning, and citizen science. *Classical and Quantum Gravity* 34, 6 (Feb 2017), 064003. https://doi.org/10.1088/1361-6382/aa5cea