

Proposal for FY2022 Laboratory Directed Research and Development Funds At Jefferson Lab

|  |  |
| --- | --- |
| **Title** | Running Legacy Code on Heterogeneous Hardware via Surrogate Models |
| **Topical area** | Group 1: High Performance Computing and Machine Learning |
| **Lead Investigator** | Nathan Brei |
| **Email:** | nbrei@jlab.org |
| **Date:** | 24 May 2021 |
| **Home Division:** | CST |
| **Other Personnel:** | David Lawrence, Kishansingh Rajput, Malachi Schram |
| **Project duration:** | **Start:** FY2022**Complete:** FY2023 |

|  |  |
| --- | --- |
| **Division Budget Analyst** | Annie Ungaro |
| **Email:** | atran@jlab.org |

This document and the material and data contained herein were developed under the sponsorship of the United States Government. Neither the United States nor the Department of Energy, nor the Thomas Jefferson National Accelerator Facility, nor their employees, makes any warranty, express or implied, or assumes any liability or responsibility for accuracy, completeness or usefulness of any information, apparatus, product or process disclosed, or represents that its use will not infringe privately owned rights. Mention of any product, its manufacturer, or suppliers shall not, nor it is intended to imply approval, disapproval, or fitness for any particular use. A royalty-free, non-exclusive right to use and disseminate same for any purpose whatsoever, is expressly reserved to the United States and the Thomas Jefferson National Accelerator Facility.

# Summary of Proposal

## Description of Project

We propose to develop tools to automatically generate machine learning surrogate models from existing software so that they may utilize modern heterogeneous hardware. A hypothetical future high performance data facility would make extensive use of heterogeneous hardware such as GPUs and FPGAs, but many legacy codes will need to be heavily modified in order to take advantage of this hardware. Surrogate models replace a piece of code which is expensive to run with an approximate model; when the underlying model is a neural net, it runs efficiently on heterogeneous hardware. Thus, they are a promising technique for offloading computation to such hardware while minimizing the necessary changes to the original code. The tools developed during this project would make it substantially simpler to implement a surrogate model, enabling legacy code to access heterogeneous hardware, saving users' time and effort, and eliminating redundant work. Lowering these barriers should enable faster development iterations and make it easier to bring machine learning research code into production. This project includes a proof of principle of a new kind of code analysis tool which could be useful for an even broader variety of problems in high performance computing. Several of the milestones open up opportunities for future research into neural differential equations and automatic identification of functions to surrogate.

## Expected Results for FY22

The first year of this project will produce, as deliverables, two software artifacts and a document. The first artifact is a toolkit for “surrogating” existing functions. This toolkit provides the user with an interface for specifying the input variables, their bounds, and the model parameters. Meanwhile it abstracts away the work of integrating TensorFlow, capturing training samples, choosing additional training samples, training the model, loading and storing the trained model, and switching between the original function and the surrogate. This toolkit serves as a bridge between idealized ML research problems and unwieldy legacy codebases.

The second artifact, the “vacuum tool”, is a program which analyzes a legacy codebase, identifies all of the target function’s inputs and outputs (this includes global variables, nested structures, side effects, etc.), and attempts to generate code that repackages them for the machine learning model. This program serves two purposes: firstly, it helps eliminate functions from consideration if their input or output spaces are too complex or they are too memory-bound; secondly, it saves the user from having to laboriously identify all of the inputs and outputs and write the code to repackage them.

The first year of this project will also produce a “playbook” document, which describes, step-by-step, how to identify compute kernels which might benefit from offloading onto heterogeneous hardware. It will discuss which profiling tools to use and what metrics to look for. This knowledge will be obtained by analyzing existing jobs on JLab’s cluster and applying the surrogate toolkit, which will be made available for future researchers.

## Funds requested for FY22

## The first year request is $196k.

# Proposal Narrative

## Purpose/Goals

High Performance Computing (HPC) facilities have moved towards heterogeneous hardware designs, which combine CPUs, GPUs, FPGAs, and sometimes TPUs.  For instance, phase 1 of the new Perlmutter system at NERSC provides 3.9 PFLOPS on CPUs, compared to at least 59.9 PFLOPS on GPUs. [9] However, the majority of existing scientific software, including reconstruction/analysis software in nuclear physics, has been developed for CPUs exclusively. To utilize heterogeneous hardware, existing codes generally need to be rewritten, costing considerable effort. One particularly promising approach is to selectively replace compute-intensive parts of the code with an approximation called a *surrogate model*, that usesML techniques under the hood. ML models run efficiently on heterogeneous hardware, and ML frameworks, such as TensorFlow, provide portability across hardware platforms. Put together, a machine-learned surrogate model requires significantly less code to be changed, uses heterogeneous hardware well, and it keeps the hardware relatively transparent to the programmer.

Surrogate models are widely used in scientific computing, particularly to explore the design space of a slow simulation. [1][2][3] The key idea behind surrogate models is to wrap a function call with a proxy that delegates to a mathematical model. Neural nets are an effective choice of model since they can be good non-linear function approximators and their weights can be dynamically updated. For training data, the proxy may capture the inputs passed to the proxy, run the original function, and capture its output. Alternatively, it may adaptively sample the original function’s input space as needed to keep the error within a specified tolerance. A variety of sampling algorithms exist, ranging from a basic grid search to Bayesian optimization.

ML models trained from an existing function have several advantages over ML models trained from data. Because they have access to the original function, they can calculate their own accuracy. They are free to generate new training samples on the fly, which reduces problems stemming from training data being out-of-distribution from test data. Finally, the surrogate model is usually differentiable, regardless of whether the original function is, which is highly useful for optimization and some ML architectures. [3][4]

However, there are several challenges which limit the use of surrogate models presently. Existing surrogate model libraries accept an arbitrary function, but the less *pure* the function is, the less likely the surrogate will be accurate*.* Purity means thatthe function only depends on its arguments (e.g., no non-local or static variables), it is referentially transparent (always returns the same output for each input), and it has no side effects (i.e. the only output is its return value, and it does not mutate anything outside itself.) [5] These libraries also assume that the input space has a simple, highly structured form, such as an n-tuple of floats. Legacy scientific codebases are practically never like this. In the case of nuclear physics reconstruction code, hidden inputs often come in the form of parameters, calibration constants, magnetic field maps, detector geometries, etc. In a similar vein, data is commonly structured in rich object trees or “awkward-arrays” [6]. Employing a surrogate model, or any machine learning model, in this context requires refactoring.

Refactoring the legacy code is a significant investment due to a variety of complications. Firstly, the compute kernels that will yield the most benefit are also likely to be the oldest and most heavily optimized. As such code often becomes more tightly coupled over time, these kernels will accumulate extra inputs and outputs compared to a textbook version of the algorithm. The extra variables are likely to be buried under nested function calls and data structures. Once identified, the variables need to be repackaged into a buffer that can be passed to the machine learning or surrogating framework, which requires tens or hundreds of lines of boilerplate. Identifying the variables is not quite enough; determining their bounds is also required. Next, a heavy framework such as TensorFlow needs to be added to the build system and integrated. Finally, there is a lurking temptation to write the sampling algorithm from scratch.

This project envisions a much simpler workflow. A researcher runs a tool such as perf or VTune to identify compute hotspots, and then runs a new tool which reports the true inputs and outputs for each function in the call stack for each hotspot. The researcher chooses which function to surrogate by examining which function has the cleanest input and output spaces, and the least memory movement. A tool then generates C++ code for packaging the inputs and outputs, training the model, and delegating to its surrogate. The researcher merely needs to copy this code into the legacy codebase, add a library dependency, set up the model, and possibly provide bounds information for some of the variables.

With this simpler workflow, researchers can iterate on their designs much faster. They are empowered to experiment with surrogates for a multitude of functions, whereas before they may have only had time for one or two. They might start tackling problems with more complex input spaces than before. Finally, they can focus on the design of their model, instead of the low-level software engineering.

## Anticipated Outcomes/Results

Fundamentally, this project seeks to answer the following four questions.

1. How do we identify compute kernels which would benefit from offloading onto heterogeneous hardware?

It may not be obvious which parts of the code can or should be offloaded, and the analysis needed to decide is an investment in its own right. The goal is to bring down the learning curve, and hence cost, of doing the analysis by formalizing it into a "playbook". This playbook would be written during the first year and updated continuously thereafter as JLab’s expertise in this area grows. If the playbook is sufficiently unambiguous, it could later become the basis of an automated discovery tool. Automated or not, JLab staff and users with legacy software would save time and effort, and arrive at a better decision about which functions to target.

1. Given a compute kernel, how do we train and incorporate a surrogate model?

Surrogate models are a promising and very general avenue for introducing machine learning into a legacy codebase, to the extent that many JLab ML researchers are likely to use one at some point. For instance, another LDRD currently proposed [7] makes use of a surrogate model, and would benefit from having a toolkit such as this.

However, implementing a surrogate model requires adding heavyweight dependencies (e.g. TensorFlow) and can be approached with varied levels of abstraction. The aim is to learn how to implement one cleanly and then encapsulate this knowledge into a well-designed and general toolkit, so that JLab staff can avoid reinventing the wheel. The milestones lay out a path to achieving this by the end of the first year.

1. How feasible is it to create a tool for extracting the 'true' inputs and outputs of an impure C++ function, and what would such a tool enable?

This is an open question. A tool like the “vacuum tool” is not known by us to exist, and it is possible that there are technical limitations preventing its existence. However, the rewards are high, and it would have utility even in a limited form. This project is designed so that if the vacuum tool doesn't succeed, the other deliverables are not affected.

If the vacuum tool does succeed, it dramatically reduces the labor needed to implement a surrogate, paving the way for rapid experimentation with using surrogates at different places in the call stack. It also provides a detailed report on the function's memory movement, which would improve the profiling playbook discussed in (1).

This tool opens up other exciting directions to investigate, as there are many things which can be done with pure functions that cannot be done with impure ones. Firstly, it can be used for generalized parameter tuning within codebases which were not designed with that in mind. By choosing a subspace of the inputs as "parameters" and sampling just those (while keeping the other inputs fixed), one can build a response surface and apply an optimization algorithm such as stochastic gradient descent to find the best parameter values. This could have applications for calibration or model fitting.

Secondly, it would enable a global sensitivity (or forward error) analysis of arbitrary scientific code. In a nutshell, one can apply small perturbations in different directions around a point in the input space, and measure the response. This could be useful for detecting numerical problems due to floating point roundoff errors.

Thirdly, it would open up additional options for testing legacy codebases. Property-based testing tools such as QuickCheck [8] traverse the input space, generating test cases and asserting that the function being tested preserves a set of invariants for each test case. This is widely used in functional programming, but not in scientific programming, because the functions to be tested are rarely pure enough. The vacuum tool may provide the missing link.

The question of whether the vacuum tool is feasible will be answered within the first year, and if so, a proof of principle will have been implemented. Making it broadly usable is expected to take at least another year, although there are too many unknowns to make detailed plans beyond the first year.

1. Which AI/ML models are effective for charged particle tracking, and how well does the surrogate perform compared to other approaches?

One of the most significant and CPU-intensive pieces of code in experimental nuclear physics is charged particle tracking. It is therefore important to ensure that the surrogate library and vacuum tool are suited for handling this task early on. This helps validate that those tools will indeed be usable by later researchers at JLab and elsewhere.

In the first year, the project builds on and extends work started by David L., Kishan R. and Dennis F., to build a ML model capable of replacing the GlueX tracking code by generating the state vectors and covariance matrices from raw hit information. It is also an opportunity to do a preliminary investigation into a different approach to tracking, involving neural ODEs (ordinary differential equations). If this approach bears fruit, it would open up further investigations in subsequent years. Either way, after the first year the focus will move away from developing the tooling and more towards applying the tooling to problems.

## Approach/Methods

The general approach to developing the code is to repeatedly build off of a simplified version of the problem. For each milestone, a subset of features is identified and a model problem is chosen which exercises just those features. These model problems are designed to be simple, well-understood, and realistic. By getting the system to work end-to-end for each model problem, the software is tested and the concept is validated in a tight feedback loop.

There are four separate threads of work:

* 1. Developing the surrogate toolkit
	2. Developing the vacuum tool
	3. Designing machine learning models used by the surrogate
	4. Writing the playbook for analyzing legacy code

While work on the playbook is independent from the other three (at least the first year, as it may ultimately inform improvements to the vacuum tool), the first three work threads have an interesting interplay. The first step is always to extend the surrogate toolkit to support the model problem, without using the vacuum tool. Once the surrogate is in place, development of the underlying ML model begins. Meanwhile, the vacuum tool is extended so that it automatically generates the code which had been manually written. The end result is that the surrogate toolkit, the vacuum tool, and the ML model all work together at the level of complexity needed to solve that model problem.

There are three model problems that will guide development in the first year:

1. Magnetic Field Map. This is chosen because its inputs and outputs are tuples of primitives. Furthermore, it has no dependencies even in complex codebases such as GlueX, and hence represents a best-case scenario. It allows an early demonstration of a surrogate model working end-to-end inside a legacy codebase. It may be possible to leverage previous JLab research to avoid designing a ML model from scratch. An added benefit is that this is a logical starting point for investigating a new approach to machine-learned tracking which uses neural ODEs to encode the system equations into a neural net explicitly. [10]
2. 2D diffusion equation. This is chosen because its inputs and outputs are grid-structured data. This is a very familiar model problem in scientific computing outside of nuclear physics, so it demonstrates that this project is useful for a broader user base.
3. Charged particle tracking. This is chosen because its inputs and outputs are rich object trees. Realistically, this is where the larger compute kernels in nuclear physics are likely to be. This builds off of JLab research that was already begun before the surrogate toolkit.

The plan for the second year is left vague because there are too many unknown unknowns at this point. The surrogate toolkit should be already be usable, but would benefit from more sophisticated sampling algorithms such as Bayesian optimization. It may also benefit from handling tuning parameters differently from other inputs. With the vacuum tool, the major remaining challenge is handling complex outputs, e.g. object trees that aren’t iterable or default constructible, although additional loose ends will likely arise. The machine-learned charged particle tracking should be developed further. The neural ODE approach to tracking should be explored as well. Depending on the lessons learned from writing the playbook, the vacuum tool may be extended to report information about memory movement that can be used to better decide which functions to surrogate. It may even be possible to further extend the tool so that it automatically identifies which functions should be surrogated.

**Goals for FY2022**

* Quarter 1
	+ Surrogate library: Wrap TensorFlow, implement a training algorithm, and create an interface for specifying the inputs, supporting primitive inputs only. Validate using magnetic field map model problem.
	+ Vacuum tool: Begin exploratory development
* Quarter 2
	+ Surrogate library: Extend as needed to support diffusion equation model problem, e.g. array inputs and outputs.
	+ Vacuum tool: Develop support for pure functions of primitives. Validate using magnetic field map model problem.
	+ ML: Continue existing research on charged particle tracking via ML, using simulation data.
	+ Playbook: Identify compute kernels in current jobs which may benefit from surrogate models
* Quarter 3
	+ Surrogate library: Extend as needed to support tracking model problem, e.g. nested object inputs and outputs.
	+ Vacuum tool: Develop support for impure functions of primitives and arrays. Validate using the diffusion equation model problem.
	+ ML: Design a model for the diffusion equation problem
* Quarter 4
	+ Vacuum tool: Develop support for nested pointer structures. Validate using tracking model problem.
	+ ML: Integrate charged particle tracking model from Q2 with surrogate library and apply it to real-world data.
	+ Playbook: Profile and analyze previously identified compute kernels; write document.



Fig 1: Dependency graph of project goals.

Different colors denote the different threads of work.

## Required Resources

This project will require modest computing time on the SciComp cluster for the purpose of profiling legacy code, distributed evenly throughout the year. It will also require computing time on GPU-enabled nodes for training the ML models. This will be readily available from JLab SciComp.

## Accomplishments in Previous Years

This is a new project.

# Budget Explanation

In terms of personnel, Nathan Brei will be spending 50% of his time this upcoming year on this project, as both the PI and the main C++ developer. Kishan Rajput will be spending 15% of his time designing machine learning models for this project. David Lawrence will be spending 10% of his time providing advice, identifying opportunities to use surrogate models in existing JLab jobs, and writing the “playbook” for future users.

A new hire (either a postdoc or SCS-I) contributes 50%, focusing on C++ software development. Ideally, they would have experience with C++ and compiler design. Because this project needs to be approved before the job can be posted, and because hiring has a lead time of at least 3 months, we expect to hire by January. The milestones are designed such that work will begin with existing JLab staff members and tasks requiring the new hire are scheduled for later. The new hire will be coordinated with other funded projects and their term will not exceed a time for which funding has not already been secured.

In terms of equipment, a computer will be purchased for the post-doc. The team anticipates being about to borrow a graphics card and FPGA; however, purchase of same might be needed (funds have not been included in the project request at this time).

Nathan Brei - 50% FTE

Kishansingh Rajput - 15% FTE

David Lawrence - 10% FTE

New hire (postdoc or SCS-I level) - 50% FTE

References

[1] Mohamed Amine Bouhlel, John T. Hwang, Nathalie Bartoli, Remi Lafage, Joseph Morlier, and Joaquim R. R. A. Martins. A python surrogate modeling framework with derivatives. *Advances in Engineering Software*, page 102-662, 2019. doi:[10.1016/j.advengsoft.2019.03.005.1](https://doi.org/10.1016/j.advengsoft.2019.03.005.1)

[2] Shannon Brown, Laura Swiler, Michael Eldred, Eric Cyr, Anthony Giunta, and Mark Richards. The surfpack software library for surrogate modeling of sparse irregularly spaced multidimensional data. 3, September 2006. doi:[10.2514/6.2006-7049](https://doi.org/10.2514/6.2006-7049)

[3] Surrogates.jl documentation. <https://surrogates.sciml.ai/latest/> Accessed: 2021-05-30.

[4] Mike Innes, Alan Edelman, Keno Fischer, Chris Rackauckas, Elliot Saba, Viral B Shah, and Will Tebbutt. A differentiable programming system to bridge machine learning and scientific computing, 2019. URL: [https://arxiv.org/abs/1907.07587v2,arXiv:1907.07587](https://arxiv.org/abs/1907.07587v2%2CarXiv%3A1907.07587)

[5] Ivan Cukic. Functional Programming in C++. Manning, January 2018. URL: <https://www.manning.com/books/functional-programming-in-c-plus-plus>

[6] Jim Pivarski, Peter Elmer, and David Lange. Awkward arrays in python,c++, and numba. *EPJ Web of Conferences*, 245:05023, 2020. doi:[10.1051/epjconf/202024505023](https://doi.org/10.1051/epjconf/202024505023).

[7] Kishansingh Rajput, Malachi Schram, Matt Bickley, Jay Benesch, and He Zhang. Multi-objective optimization of CEBAF heat load management and trip rates using AI/ML. LDRD proposal, 2021.

[8] Koen Claessen and John Hughes. Quickcheck: A lightweight tool for random testing of haskell programs. *SIGPLAN Not.*, 35(9):268–279, September 2000. doi:[10.1145/357766.351266](https://doi.org/10.1145/357766.351266)

[9] NERSC documentation: System details, phase 1. <https://docs.nersc.gov/systems/perlmutter/system_details/> Accessed: 2021-05-30.

[10] Ricky T. Q. Chen, Yulia Rubanova, Jesse Bettencourt, and David K Du-venaud. Neural ordinary differential equations. In S. Bengio, H. Wallach,H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 31. Curran Associates, Inc., 2018. URL: <https://proceedings.neurips.cc/paper/2018/file/69386f6bb1dfed68692a24c8686939b9-Paper.pdf>