Tmedi: Tumor Microenvironment Data Integration

I Executive Summary

Linguistic Technology Systems (LTS) is working on a novel suite of software tools supporting Tumor Microenvironment (TME) modeling, an area of research in Precision Medicine that has gained appreciable traction in recent years.

Such data-management and data-integration tools, which we refer to as *Tmedi* (Tumor Microenvironment Data Integration), are especially designed for contemporary bioinformatics computing environments that feature heterogeneous, multi-disciplinary data profiles as well as diversity in data-management architectures: e.g. workflows that use different data-access methods. They also meet some very important challenges elucidated by major research panels and organizations. For example, the *National Center for Multiscale Modeling of Biological Systems* (MMBIOS), in a recent statement espousing both research and computational priorities, acknowledged the challenges of data integration and the importance of a common data model covering a broad spectrum of immuno-oncology and systems-biology research. The goals cited by MMBIOS in the past year include "Subaim 2.2: Develop a database of molecules, rules, and models that can be used for comparative analysis of existing models and development of new models" and "Subaim 3.1: Develop a hardened C/C++ library and application programming interface (API) for BioNetGen that exposes both high- and low-level functionality."[1]

To address these concerns, LTS proposes to implement C++ database engineering and runtime data modeling libraries which could be used toward the completion of these aims and/or toward integrating the code libraries — that have been created in response to the priorities of MMBIOS — into new and/or existing biomedical applications.

II Background

LTS recognizes that biomedical database technology has seen a broad diversification over the last decade, with augmented forms of graph databases (such as property graphs and hypergraphs), matrix-based

[1] These aims are declared at https://mmbios.pitt.edu/research/technology-research-and-development/network-modeling.

databases, and other distinct branches of database engineering systems which play a significant role in clinical, research, and translational medicine. Moreover, large quantities of clinical, research, and diagnostic information are increasingly being stored in heterogeneous data pools which tend to be larger and more general-purpose than individual databases.

For example, different institutes or departments within a research hospital may all persist information in a common data "lake," serving two purposes:

1. Implementing common requirements, data accession, security, depersonalization, version control, and other logistical concerns which are orthogonal to data structures themselves (even if the data produced by each department is unique, requiring its own data models and applications); and

2. Centralizing data in a common store so as to afford researchers the means to design queries, applications, or analytic modules (e.g., machine learning algorithms) that sample parameters derived from a range of data profiles with distinct provenance.

For bioinformatics, the consequence of such developments, mentioned above, namely the diversification in database architectures and the generalization of databases to data pools and "lakes," is that applications and software components which export data for data sharing and data persistence need to anticipate a wide range of contexts where this data may be stored and used. Conventional application-to-database integration methods (such as Object-Relational Mapping, Object-Triple-Mapping, or code-generated SQL queries) are inadequate for contemporary technology, because one cannot assume that application data will be exported to a storage layer physically organized as a conventional relational database or a triple-store, or any other specific kind of data-management back-end.

Although heterogeneous data lakes have received industry attention as warehouses for clinical and diagnostic data, these trends are indeed directly relevant to those producing and curating research data as well. This is so, first because clinical and diagnostic data is often an important source for research data (providing case-studies for analysis, training sets for Machine Learning, empirical parameters for simulation models, etc.) and second, because researchers are often encouraged to deposit data in shared archives similar to data lakes.

To address these concerns, LTS proposes to implement a database engine which responds to the **MMBioS** aims; and in particular to optimize the database for research where data is accessed, shared, and deposited according to contemporary data-management paradigms (such as data lakes and various **NoSQL** architectures) in contrast to conventional research data management (which tends to exhibit characteristics reflecting older norms such as relational database models).

III Toward a Shared Research-Oriented Software Development Environment for TME

The MMBioS projects prioritize data integration because of the architectural paradigm governing how MMBioS research is organized — specifically, four distinct "Technology Research and Development" domains, including Molecular Modeling, Cell Modeling, Network Modeling, and Image Processing. While these domains focus on different scales of biochemical systems and different analytic tools, MMBioS prioritizes integration among software built for each domain; for instance, emphasizing the need for "interoperability of TR&D1 tools with the major software ... being developed in the respective TR&Ds 2-4 toward building a computational platform for integrated structural cell biology."

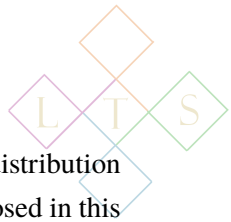
Because TME and systems-biology research is inherently inter-disciplinary, particular research initiatives — for example, the four MMBioS TR&D domains — will usually draw data and code from multiple different sources. For example, a TME simulation may be modeled via initial parameters whose ranges are established by mining both genomic data (derived from sequence alignment algorithms) and proteomics (derived from cytometry).[2] Similarly, mathematical models may be developed and tested against data from a range of different sources, such as genomic, biomolecular, and epidemiological data,[3] or may combine equations originating in one field of research with empirical data mined from other fields.[4] In these scenarios, research methods — and sometimes computer code — are often included in "Supplemental Materials" sections or "Data Availability" statements attached to publications. Such supplemental material documents the steps which researchers have taken to acquire and analyze their data, and clarifies their research/computational methods.

However, research which spans multiple data sources and computing environments is intrinsically difficult to reproduce, even if data sources and computational methods are clearly described. While research environments in some contexts, such as data science, can be digitally "packaged" using technologies such as Kaggle or Jupyter, the implementation of "digital notebooks" is more difficult in heterogeneous fields such as TME research. New technology is therefore needed to support shareable research in contexts such as the MMBioS domains. For example, simulating the geometric evolution of a tumor (using local

[2] See for instance Mohammad Jafarnejad, *et. al.*, "Mechanistically detailed systems biology modeling of the HGF/Met pathway in hepatocellular carcinoma." This paper in particular describes a multi-part research workflow combining genomic data — specifically, Cancer Genomic Atlas (TCGA) sequences — with cellular data (from the Broad Institute Cancer Cell Line Encyclopedia), with protein abundance metrics derived via mass cytometry, and with simulations of intracellular signaling and of therapy regimens targeting these signaling mechanisms.

[3] Steffen Klamt, Utz-Uwe Haus, and Fabian Theis, "Hypergraphs and Cellular Networks" examines hypergraph-analysis methods applied to several datasets and mathematical models relevant to systems biology, such as genetic regulatory networks, human-disease networks, and protein complexes.

[4] For instance, Oleg Milberg, *et. al.*, "A QSP Model for Predicting Clinical Responses to Monotherapy, Combination and Sequential Therapy Following CTLA-4, PD-1, and PD-L1 Checkpoint Blockade." This paper describes the process of constructing a "virtual patient" cohort by mining real-world clinical data and then using this collection of virtual-patient profiles as the basis for a Quantitative Systems Pharmacology (QSP) model simulating a tumor microenvironment via biochemical equations, a simulation which propagates to a model of tumor evolution that can be used to investigate immunotherapy mechanisms.



evolution rules) in conjunction with the clinical data on tumor growth (establishing statistical distribution for how tumors evolve empirically) would be made feasible by using tools, such as those proposed in this White Paper, to integrate clinical data with simulation code.

LTS proposes to address these challenges of sharing research data and computational capabilities in such heterogeneous areas by formulating a software development environment where custom native/desktop applications, encapsulating individual research projects, could be implemented and reused. Such dataset-specific applications would be self-documenting and shareable in the manner of a Jupyter notebook, but would be rendered more flexible, thereby giving developers control over GUI design and data-export procedures. Each such software application specific to a given research project could thereby draw from a common collection of code libraries and development tools, so that the application could be shared between researchers who download the *Tmedi* environment. Moreover, project-specific applications could be configured to acquire and export data to/from a common data model and database engine designed for TME research, based on existing initiatives in this subject-area such as MMBIOS, the Poppel Systems Biology Laboratory, the Institute for Systems Biology, and the Englander Institute for Precision Medicine at Weill Cornell Medicine/Weill Cornell Graduate School of Medical Sciences.

IV Understanding the Challenges

The challenges of integrating heterogeneous data sources are especially pronounced in research areas such as systems biology and tumor microenvironment modeling, both of which are at the forefront of contemporary oncology and cancer immunotherapy. This is so because such research must frequently combine disparate investigative modalities, such as image analysis, flow or mass cytometry, confocal microscopy, genomics, 3D geometry, and *in silico* simulations of biological systems. In practice, such variegated data sources are often integrated for each particular research project in *ad-hoc* ways, which albeit sufficient for publishing individual research papers is less useful for those seeking to replicate, reuse, or refine research work after it has been published. This problem is exacerbated by a diversity of programming languages and environments used to produce code which backs up research in systems biology and immunotherapy. In particular, Matlab, Python, R, C++, and Java, though all widely used in these areas, all carry their own ecosystem of tools and libraries which make these languages difficult, if not impossible, to interoperate.

Additional code-fragmentation becomes evident once we consider data visualization as well as data acquisition and analysis. To be specific, it is not uncommon for research tools (such as **BioConductor**) or individual research projects to employ one language (particularly C/C++) for data processing but an entirely different language to build user-visible GUIs.[5] Overall, as a cumulative result of implementations

[5] For instance, in Chang Gong, *et. al.*, "A computational multiscale agent-based model for simulating spatio-temporal tumour immune response to PD1 and PDL1 inhibition," a lattice-based geometric simulation of tumor growth in reaction to the tumor microenvironment is implemented in C++ but paired with front-end code in the Tk language.

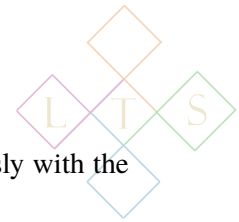
branching out in different directions at different coding stages (such as data acquisition, analytics, and visualization), programming requirements are split among a plethora of isolated platforms rather than being centralized in a common software development environment. This situation adversely impacts data integration, because it is much more difficult to integrate data which originates from disparate programming environments.

V Meeting the Challenges

To address the challenges raised by heterogeneous data profiles in areas such as TME research, LTS proposes to develop a database engine in the context of a multi-faceted development environment. Such an initiative, applicable to the database and the C++ library aims of MMBIOS as part of an integrated platform for TME, systems biology, and immuno-oncology research, would be focused on building and extending software applications used for the more encompassing systems-biology/immunotherapy research platform. As such, this platform would bundle a number of important C++ libraries currently used for research in these areas, such as LIBSBML (a parser for Systems Biology Markup Language), MSIsensor (which supports next-generation sequencing for tumor genomes), CYTOLIB (for reading flow cytometry files), CAPTK (the Cancer Imaging Phenomics Toolkit, for cancer-related image processing) and PhysiCell (for 3D simulations of cellular systems), together with a sizeable collection of general-purpose bioinformatics and/or scientific computing libraries (Eigen, Armadillo, BFP, DCMTK, Common Workflow Language, WhiteDB, TileDB, Cap'nProto, DeltaQt, Meshlab, VXL, Gamgee, IQMOL, Paraview, EpiFire, etc.).

LTS would integrate these libraries/applications in a common development environment intended to facilitate inter-operability while also minimizing external dependencies, modifying the libraries as necessary to achieve this goal. For example, LTS has previously developed an alternative version of CYTOLIB which eliminated dependencies on Protocol Buffers and on R packages from **BioConductor**. Within the context of this development environment, LTS would provide capabilities for declaring and executing workflows spanning multiple components, allowing research workflows in the focal areas of TME, systems biology, and cancer immunotherapy to be conveniently tested and replicated. Moreover, LTS would provide data-export protocols that could be used to pool research data into a common data model, adding support for such protocols as plugins to components included in the development environment. Finally, LTS would provide an integrated query system combining features of property graphs, hypergraphs, and data-lake models to retrieve information from the proposed MMBIOS database and/or from components included in the development environment proposed here.

Consistent with the goal of instantiating a database engine and a concrete database of models, molecules, and simulation rules for systems biology, LTS proposes a hybrid property-hypergraph model which incorporates technologies associated with property graph, hypergraph, and data-lake style information warehousing. Currently, both property-graph and hypergraph engines have been implemented primarily in the context of Java. However, implementing a combined property-hypergraph system in C++ allows state-of-



the-art graph-database designs to be deployed in an environment that interoperates seamlessly with the vast ecosystem of scientific/bioinformatics applications and code libraries written in C++.

Commensurate with property graphs, LTS's combined property-hypergraph model allows named properties to be associated with any nodes, edges, or parts of a graph for the sake of conveniently retrieving information and/or traversing graph instances. Moreover, the property-hypergraph model supports graph traversal algorithms described via the Gremlin Virtual Machine, which LTS can port to C++ as part of the MMBIOS implementation. At the same time, consistent with hypergraphs, the property-hypergraph model supports arbitrary value-cardinality, hyperedges with three or more incident nodes (potentially labeled via "roles" as in **Grakn.ai**), and structures for aggregating related hypernodes and hyperedges into frames and contexts. Hypernodes in this model can represent arbitrarily large data collections with predetermined rules for adding or removing values (such as sorted lists, Last-in-First-out stacks, and First-in-First-out queues). The proposed database engine would use memory layout optimized to support efficient modification and querying of collections-style hypernodes and multiple-cardinality data values.

VI Introducing the LTS "Property-Hypergraph Virtual Machine (P*h*VM)"

In order to add information to and retrieve information from the proposed database, the LTS property-hypergraph hybrid has introduced a new C++ Virtual Machine, dubbed "Property-Hypergraph Virtual Machine" (P*h*VM). One goal of P*h*VM is to provide C++ support for the Gremlin VM (used to query property-graphs) as mentioned above. Indeed, by compiling to a Virtual Machine, this framework can support property-graph traversal via textual queries rather than (as is currently the case given how Gremlin VM is employed) solely within application-level source code. The scope of P*h*VM is much broader, however.

To facilitate implementation of middle-tier components connecting applications to property-hypergraphs, P*h*VM would support special scripting or Domain-Specific languages (DSLs) describing how application-level data types should be serialized for export to a database or common data pool. In P*h*VM, *querying* and *serialization* are seen as interrelated operations, essentially inversions of one another: serialization encodes all information contained within a typed value so as to store this data in a persistent state; whereas querying locates a particular typed value (potentially out of many alternatives in a large data base) and recreates the application-level value by extracting whatever data-points are needed from the data base. These operations are *de facto* inverses insofar as each value may be serialized in a manner that anticipates queries which will later be used to find and reconstruct the value (which in turn requires a more detailed serialization model than that provided by application-level libraries such as, in the C++ context, **boost** or QT).

P*h*VM assumes that most typed values are aggregate structures encompassing multiple data fields and/or meta-data properties, some of which themselves may be handles to other aggregate structures and/or arbitrary-cardinality lists, sets, stacks, or queues. Each typed value in data storage may then be associated

with dozens or perhaps hundreds/thousands of subordinate values. Some of this secondary information need merely be encoded with the original value and then decoded when the original value is deserialized; but other secondary values would have to be indexed and reified so that they can be used as the basis of queries, as foreign-key references for table-style joins, or as routes for graph traversal operations. For example, a person's last name, street address, or Patient ID number are all reasonable pieces of information with which to identify a single person from a database of clinical and/or diagnostic records, so these data-fields would typically need to be indexed and reified for database queries. One purpose of **P*h*VM**, accordingly, is to support test scripts and documentation describing type-level annotations for such details via a run-time meta-class reflection system.

VII Configuring the P*h*VM with the QT Meta-Object System and Meta-Object Compiler (MOC)

LTS proposes to develop code applicable to the database and C++ library aims of MMBIOS as part of an integrated platform for TME, systems biology, and immuno-oncology research. Since this platform would bundle a number of important C++ libraries currently used for research in these areas, LTS would integrate these libraries/applications in a common development environment intended to facilitate interoperability while also minimizing external dependencies, modifying the libraries as necessary to achieve this goal. For example, LTS has previously developed an alternative version of CYTOLIB which eliminated dependencies on Protocol Buffers and on R packages from **BioConductor**. Within the context of this development environment, LTS would provide capabilities for declaring and executing workflows spanning multiple components, allowing research workflows in the focal areas of TME, systems biology, and cancer immunotherapy to be conveniently tested and replicated. Moreover, LTS would provide data-export protocols that could be used to pool research data into a common data model, adding support for such protocols as plugins to components included in the development environment. Finally, LTS would provide an integrated query system combining features of property graphs, hypergraphs, and similar graph-oriented data models to retrieve information from the proposed MMBIOS database and/or from component included in development environment proposed here.

Consistent with the goal of instantiating a database engine and a concrete database of models, molecules, and simulation rules for systems biology, LTS proposes a hybrid property-hypergraph model which incorporates technologies associated with property graph, hypergraph, and data-lake-sty;e information warehousing. To this point, both property-graph and hypergraph engines have been implemented primarily in the context of Java. Implementing a combined property-hypergraph system in C++ allows state-of-the-art graph database designs to be deployed in an environment that interoperates seamlessly with the large ecosystem of scientific and bioinformatic applications and code libraries written in C++. Consistent with property graphs, LTS's combined property-hypergraph model allows named properties to be associated with any nodes, edges, or parts of a graph for the sake of conveniently retrieving information

and/or traversing graph instances. Moreover, the property-hypergraph model supports graph traversal algorithms described via the Gremlin Virtual Machine, which LTS can port to C++ as part of the MMBIOS implementation. At the same time, consistent with hypergraphs, the property-hypergraph model supports arbitrary value-cardinality, hyperedges with three or more incident nodes (potentially labeled via "roles" as in **Grakn.ai**), and structures for aggregating related hypernodes and hyperedges into frames and contexts. Hypernodes in this model can represent arbitrarily large data collections with predetermined rules for adding or removing values (such as sorted lists, Last-in-First-out queues, and First-in-First-out stacks). The proposed database engine would use memory layout optimized to support efficient modification and querying of collections-style hypernodes and multiple-cardinality data values.

In order to manage information within the proposed database, the property-hypergraph hybrid would provide the option of employing PhVM, described earlier. One goal of PhVM is to provide C++ support for the Gremlin VM (used to query property-graphs) as mentioned above. Indeed, by compiling to a Virtual Machine, this framework can support property-graph traversal via textual queries rather than (as is currently the case given how Gremlin VM is employed) solely within application-level source code. The scope of PhVM is much broader, however. To facilitate implementation of middle-tier components connecting applications to property-hypergraphs, PhVM would support special scripting or Domain-Specific languages (DSLs) describing how application-level data types should be serialized for export to a database or common data pool.

The PhVM type system would by design interoperate seamlessly with the QT Meta-Object System and Meta-Object Compiler (MOC) and would also generalize the Hypergraph Type System which was originally introduced by the HYPERGRAPHDB database engine. In the context of HYPERGRAPHDB, the Hypergraph Type System is designed to route information between applications and databases; in PhVM, more broadly, data-types registered with the PhVM Type System encapsulate functionality for exporting data to a persistent data store (such as a property-hypergraph database), for sharing data via a cloud service, or for constructing GUI components to visualize particular data structures in the form of dialog windows that can be embedded in software applications. Each type in PhVM serves as an intermediate connection point tying application-level data types with database, cloud networking, and GUI representations. This architecture is consistent with the HYPERGRAPHDB Application Model, where types are "internal to" each database (types are nodes, and type information is treated as ordinary database content). The proposed MMBIOS database implementation could provide a similar architecture for representing native/desktop application-state and application elements (such as GUI components, ranging in complexity from buttons and tabs to complex windows) which interactively render database content for application users. This would enable multiple distinct applications which all employ the MMBIOS database to provide similar functionality and User Experience for those recurring elements which they have in common.

As described earlier, applications developed with the aid of *Tmedi* tools can serve as "digital notebooks" optimized for sharing data and code between researchers. This dimension of data/code reuse and sharing would be enhanced by building research applications on top of a TME-specific database engine, where

application components and state are stored directly in the database (consistent with HyperGraphDB). To summarize, the MMBioS database would serve as a common reference point for a collection of *Tmedi*-driven applications customized for individual research projects. Each application would export data in a common TME-oriented data format, so that all of the data from these various applications could potentially be merged into a single MMBioS database instance.

For data sharing and reuse to be possible on this scale, the data model native to this MMBioS database would need to be flexible enough to ingest data with any profile that may be found in research workflows or supplemental material associated with TME research papers — including genomic, cytometric, preteomic, and bioimaging (such as confocal microscopy) data as well as geometric and/or biochemical simulations of tumor growth and tumor/environment interactions. Such a project would need a custom-built database engine, because conventional databases (even NoSQL engines which are not locked in to predefined schema) are generally not flexible enough to cover this full range of data types/profiles (at least while preserving structural integrity). To address these important concerns, LTS proposes a hybrid property-hypergraph architecture along with a Qt and HyperGraphDB-compatible type system (and a structured, query-aware serialization and distributed-computing feature-set similar to Cap'nProto) to provide a novel database framework adequate for the requirements implicit in the MMBioS goals.

Appendix "A": A New Query Language for Supporting Property-Hypergraph and Data-Lake-Style Data/Knowledge Management Architectures

As a unifying element bridging different features and components of our hybrid property-hypergraph model, LTS proposes designing and engineering a novel "Transparent Hypergraph Query Language" (THQL) formulated for the unique requirements of jointly supporting property-graph, hypergraph, and data-lake information/knowledge management architectures. THQL is called "transparent" because, ideally, all query evaluation is performed via code which can be transparently examined by developers, where the query capabilities are enabled by source-code files that can be seamlessly dropped in to application projects, and where query evaluators can be examined at runtime via debugging sessions. The code libraries and development environment packaged for use in connection with the proposed database implementation would be curated to support transparency in this sense — for example, as much as possible, library dependencies would be supplied within the development environment rather than installed separately via system package managers (which are more likely to be opaque to development tools; e.g., they may not provide debug versions of their internal libraries). Moreover, THQL could model distributed-computing requests to support research environments where computationally intensive operations are delegated to external servers, so that a particular research workflow needs to integrate and synchronize local and remote operations. In the context of the proposed MMBioS database, THQL could be optimized to support queries and serialization of rules, models, and biochemical objects identified as priorities by

the MMBioS Network Modeling project.

Appendix "B": Supplementary Note on Code Libraries

As described above, LTS's proposed database implementation would be paired with a development environment which packages numerous third-party bioinformatic and scientific-computing code libraries. One rationale for integrating these diverse libraries would be to facilitate data-integration; with the source code for various libraries and applications available to developers, provided as part of a development environment which simplifies the process of building and testing application plugins/extensions, it would be easier to extend the bundled libraries/applications with data-export features conforming to the property-hypergraph protocol. A second goal would be to promote overlapping and shared/reusable components that would help tie together different research projects (much like how existing research-software projects such as **BioConductor** or **BioC++** are designed). When researchers employ code which is either based on components provided in a shared development environment, or can be reproduced via such components, it is easier to evaluate, reuse, and reproduce published research.

Of course, thousands of scientific-computing libraries are used for various research projects; any one library package can only include a small subset of components which may be relevant for a specific area (such as systems biology). Moreover, many useful tools require special computer hardware and/or special build steps which make them difficult to incorporate into a common environment whose goal is to serve as a kind of "common denominator" available across many different institutions that would have an interest in examining or replicating published research. Certain very general or ubiquitous components, such as the Visualization Toolkit (VTK), OpenCV Computer Vision library, or Qt (for GUI code) can reasonably be accepted as external dependencies, even if the overall goal of a development environment is to minimize external dependencies which are narrower in scope. Given this orientation, accordingly, the goal for this environment would be to select libraries/applications which do not require exceptional hardware or build/install operations, and which can be provided in a largely self-contained manner, with dependencies included as their own source code files. The following is a list of libraries which would be good candidates for inclusion in this project:

libSBML https://github.com/opencor/libsbml

cytoLib https://github.com/RGLab/cytolib

CaPTk https://github.com/CBICA/CaPTk

PhysiCell http://physicell.org/

Eigen https://eigen.tuxfamily.org/

Armadillo http://arma.sourceforge.net/

BFP (Beyond Floating Point, for posit/unum real-valued computing) https://github.com/libcg/bfp

DCMTK https://dcmtk.org/dcmtk.php.en

CWL (Common Workflow Language) https://www.commonwl.org/

WhiteDB http://whitedb.org/

TileDB https://tiledb.com/

Cap'n Proto https://capnproto.org/

DeltaQt http://deltaqt.sourceforge.net/

MeshLab https://www.meshlab.net/

VXL https://vxl.github.io/

IQmol http://iqmol.org/

ParaView https://www.paraview.org/

EpiFire https://github.com/tjhladish/EpiFire

Gamgee https://github.com/broadinstitute/gamgee

YATO https://github.com/agruzdev/Yato

NYTL https://github.com/nyorain/nytl

YAAL https://github.com/AmokHuginnsson/yaal

MOAB https://sigma.mcs.anl.gov/moab-library

QCustomPlot https://www.qcustomplot.com

QScintilla https://riverbankcomputing.com/software/qscintilla

AngelScript https://www.angelcode.com/angelscript

Other potential libraries, depending on their usability in multiple contexts:

PETSc (for Partial Differential Equations) https://www.mcs.anl.gov/petsc/

Chaste (Cancer, Heart And Soft Tissue Environment) https://journals.plos.org/ploscompbiol/article?\id=10.1371/journal.pcbi.1002970

SciData https://stuchalk.github.io/scidata

NCBI C++ ToolKit https://ncbi.github.io/cxx-toolkit

Qtilities https://jpnaude.github.io/Qtilities

Qwt https://qwt.sourceforge.io