

# GAME: Genomic API for Model Evaluation

Authors:

Ishika Luthra<sup>1</sup>, Satyam Priyadarshi<sup>1</sup>, Rui Guo<sup>2</sup>, Lukas Mahieu<sup>3</sup>, Niklas Kempynck<sup>3</sup>, Damion Dooley<sup>4</sup>, Dmitry Penzar<sup>6</sup>, Ilya Vorontsov<sup>6</sup>, Yilun Sheng<sup>7</sup>, Xinming Tu<sup>7</sup>, Adam Klie<sup>8</sup>, Shiron Drusinsky<sup>10,11</sup>, Alexander Floren<sup>11</sup>, Ethan Armand<sup>25</sup>, Kaur Alasoo<sup>24</sup>, Georg Seelig<sup>7,23</sup>, Ryan Tewhey<sup>22</sup>, Peter Koo<sup>21</sup>, Vikram Agarwal<sup>20</sup>, Sager Gosai<sup>19</sup>, Luca Pinello<sup>16,17,18</sup>, Michael A. White<sup>15</sup>, Avantika Lal<sup>14</sup>, Julia Zeitlinger<sup>13</sup>, Katherine S. Pollard<sup>10,11,12</sup>, Maxwell Libbrecht<sup>9</sup>, Hannah Carter<sup>8</sup>, Sara Mostafavi<sup>7</sup>, Ivan Kulakovskiy<sup>6</sup>, Will Hsiao<sup>4,5</sup>, Stein Aerts<sup>3</sup>, Jian Zhou<sup>2</sup>, Carl G. de Boer<sup>1</sup>

Correspondence should be addressed to CGD ([carl.deboer@ubc.ca](mailto:carl.deboer@ubc.ca)).

## Affiliations

<sup>1</sup>School of Biomedical Engineering, University of British Columbia, Vancouver, BC, Canada

<sup>2</sup>Lyda Hill Department of Bioinformatics, University of Texas Southwestern Medical Center, Dallas, TX, USA

<sup>3</sup>VIB Center for AI & Computational Biology, VIB-KU Leuven Center for Brain and Disease Research & KU Leuven Department of Human Genetics, Leuven, Belgium

<sup>4</sup>Centre for Infectious Disease Genomics and One Health, Simon Fraser University, Burnaby, BC, Canada

<sup>5</sup>Faculty of Health Sciences, Simon Fraser University, Burnaby, BC, Canada

<sup>6</sup>Vavilov Institute of General Genetics, Russian Academy of Sciences, 119991 Moscow, Russia

<sup>7</sup>Paul G. Allen School of Computer Science and Engineering, University of Washington, Seattle, WA

<sup>8</sup>Department of Medicine, Division of Genomics and Precision Medicine, University of California San Diego, La Jolla, CA, USA

<sup>9</sup>School of Computing Science, Simon Fraser University, Burnaby, British Columbia V51 1S6, Canada

<sup>10</sup>University of California, San Francisco, San Francisco, CA 94143, USA

<sup>11</sup>Gladstone Institute of Data Science & Biotechnology, San Francisco, CA, USA

<sup>12</sup>Chan Zuckerberg Biohub SF, San Francisco, CA, USA

<sup>13</sup>Stowers Institute for Medical Research, Kansas City, MO 64110, USA

<sup>14</sup>Biology Research | AI Development, gRED Computational Sciences, Genentech, South San Francisco, CA, USA

<sup>15</sup>Department of Genetics, Washington University in St. Louis, St. Louis, MO, 63110, USA

<sup>16</sup>Molecular Pathology Unit, Krantz Family Center for Cancer Research, Massachusetts General Hospital, Boston, MA, USA.

<sup>17</sup>Gene Regulation Observatory, The Broad Institute of Harvard and MIT, Cambridge, MA, USA.

<sup>18</sup>Department of Pathology, Harvard Medical School, Boston, MA, USA.

<sup>19</sup>Broad Institute of MIT and Harvard, Cambridge, MA, USA

<sup>20</sup>mRNA Center of Excellence, Sanofi, Waltham, MA 02451, USA

<sup>21</sup>Cold Spring Harbor Laboratory, 1 Bungtown Road, Cold Spring Harbor, NY 11724, USA

<sup>22</sup>The Jackson Laboratory, Bar Harbor, ME, USA

<sup>23</sup>Department of Electrical & Computer Engineering, University of Washington

<sup>24</sup>Institute of Computer Science, University of Tartu, Tartu, Estonia

<sup>25</sup>Integrative Biology Laboratory, Salk Institute for Biological Studies, 10010 N. Torrey Pines Road, La Jolla, CA 92037, USA

## Abstract

The rapid expansion of genomics datasets and the application of machine learning has produced sequence-to-activity genomics models with ever-expanding capabilities. However, benchmarking these models on practical applications has been challenging because individual projects evaluate their models in ad hoc ways, and there is substantial heterogeneity of both model architectures and benchmarking tasks. To address this challenge, we have created GAME, a system for large-scale, community-led standardized model benchmarking on user-defined evaluation tasks. We borrow concepts from the Application Programming Interface (API) paradigm to allow for seamless communication between pre-trained models and benchmarking tasks, ensuring consistent evaluation protocols. Because all models and benchmarks are inherently compatible in this framework, the continual addition of new models and new benchmarks is easy. We also developed a Matcher module powered by a large language model (LLM) to automate ambiguous task alignment between benchmarks and models. Containerization of these modules enhances reproducibility and facilitates the deployment of models and benchmarks across computing platforms. By focusing on predicting underlying biochemical phenomena (e.g. gene expression, open chromatin, DNA binding), we ensure that tasks remain technology-independent. We provide examples of benchmarks and models implementing this framework, and anticipate that the community will contribute their own, leading to an ever-expanding and evolving set of models and evaluation tasks. This resource will accelerate genomics research by illuminating the best models for a given task, motivating novel functional genomic benchmarks, and providing a more nuanced understanding of model abilities.

## Main

While sequence-based genomics models are widely used in the field, their progress has proven difficult to benchmark due to substantial heterogeneity in model architectures, training datasets, and ad hoc model evaluations. Presently, model evaluations are siloed within publications, resulting in duplicated efforts; simultaneous differences in model architectures, training data, and evaluation tasks often obscure where any innovation lies. In general, functional genomics assays measure the genome's biochemical activity indirectly. Consequently, some performance gains are likely attributable to the models capturing assay-specific biases better, rather than capturing the underlying biological phenomenon. Further, it is commonplace to use genomics models to predict activities that are related to but distinct from the tasks they were designed to predict (e.g. using CAGE predictions for mRNA expression even though they capture different RNA subsets)<sup>1</sup>. Finally, even if a benchmark is superficially the same, there are often many choices (degrees of freedom) in implementation that may impact the performance measure. Consequently, the overall performance of a new genomics model compared to its predecessors and contemporaries often remains opaque. Recent work showed that proper and consistent benchmarking can reveal model limitations<sup>2-6</sup>. For instance, Sasse et al (2023) showed that, while models are able to predict the magnitude of variant effects, they struggle with predicting the direction of the change in expression<sup>3</sup>. As the number of potential benchmarking datasets continues to grow, facilitating their uniform application across models becomes increasingly important.

In other fields, there are well-established, high-quality datasets that can be used to benchmark progress because it is generally agreed that they capture the task well. For instance, Critical Assessment of Structure Prediction (CASP)<sup>7</sup> is used to benchmark protein folding models, while the CIFAR datasets are standard benchmarks for image classification<sup>8</sup>. Similarly, there have been efforts to curate diverse sequence datasets into a standard format to facilitate model training and evaluation<sup>5,9</sup>. While challenges and competitions like DREAM<sup>10,11</sup> and Kaggle<sup>12</sup> are excellent at determining the optimal model design choices for specific datasets, they are, by their nature, restricted to specific tasks. However, there are many distinct but intertwined activities encoded by the genome (e.g. RNA expression, open chromatin, transcription factor binding, histone marks, 3D conformation) that differ across species, cell types, and conditions, making it necessary for sequence-based genomics models to be benchmarked on tasks that match the model's scope and purpose. Further, users may want to identify the best model for their specific task of interest.

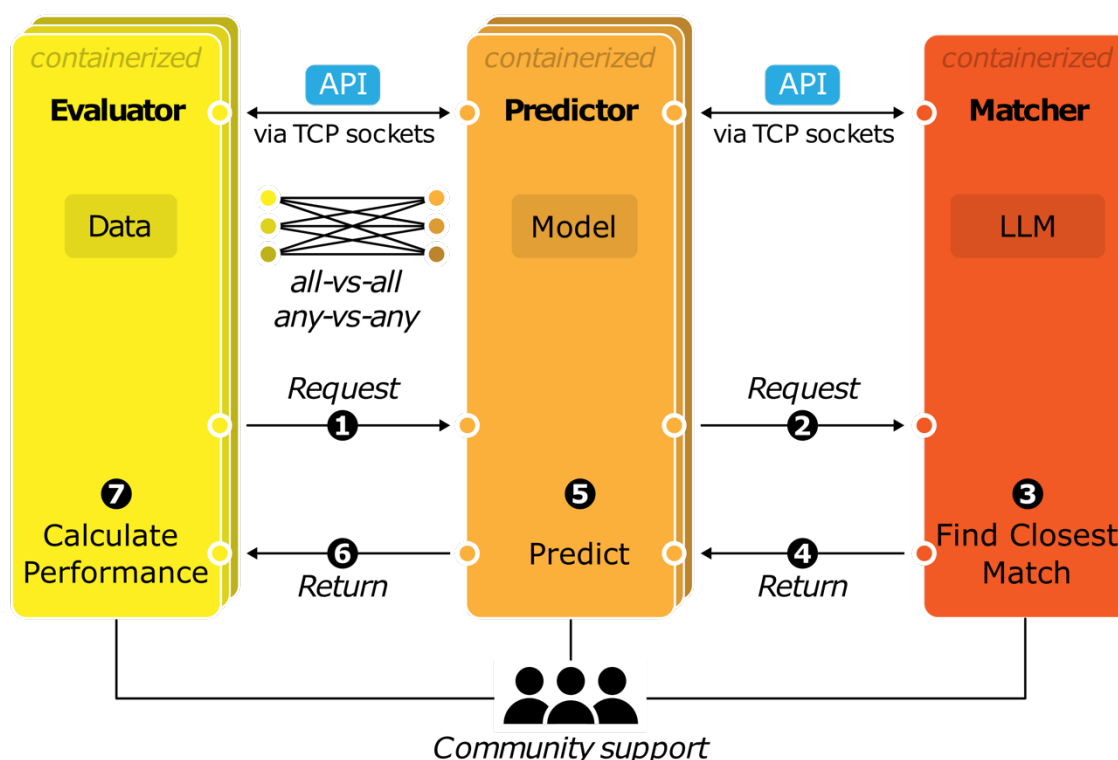
To enable systematic benchmarks of various models by various evaluation tasks, we created **GAME (Genomics API for Model Evaluation)**, a framework for large-scale community-led standardized model benchmarking. GAME is designed to be flexible and user-friendly, where new and existing models can be systematically and uniformly queried with an ever-expanding variety of tasks. To facilitate this uniform benchmarking of models across tasks, we leveraged the standardized communication protocols used by APIs<sup>13,14</sup>. APIs enable diverse client programs to interact with server programs, each without knowing any of the details of the other's implementation, facilitated by the common language defined by the API protocol. While Avsec et al. (2019) created a repository, Kipoi, for genomics-based ML models and also used APIs to access them, it did not include benchmark modules, and it required the user to have intimate knowledge about exactly what each model was predicting, making scalable, uniform model benchmarking a substantial challenge<sup>15</sup>. GAME allows for seamless communication between pre-trained models and functional genomics datasets on which those models can be benchmarked. GAME was designed with extensive community feedback throughout its development, resulting in API specifications that meet the needs of model and dataset experts.

To maximize GAME's utility and adaptability, we established several forward-looking design principles that emphasize creating benchmarks that are as objective and biology-focused as possible. Firstly, we focused on the biological signals of interest, not the assay used to quantify them. Accordingly, assay types are not provided to the models, which could use this information to improve performance but in biologically meaningless ways (e.g. adding assay-specific biases). For example, DNase I-seq and ATAC-seq both measure chromatin accessibility in different ways and have different biases. While predicting the assay-specific biases would increase performance, it provides no insight into the underlying biology. This has the secondary benefit of being robust to the development of new assays, which are likely to measure the same underlying biological phenomena in different ways. For instance, the recently-developed Fiber-seq also measures chromatin accessibility, but the nature of the data and biases differ substantially from DNase I-seq and ATAC-seq<sup>16</sup>. Accordingly, we restrict the predictions to DNA binding (which can be for TFs, methylation, or histone marks), chromatin accessibility, 3D-chromatin conformation, and expression (mRNA, Pol I, Pol II, Pol III). Secondly, we aimed to provide the utmost separation between the prediction and evaluation of these models by providing models only as much information as needed to perform their tasks and no more to prevent overfitting to the benchmarks. For instance,

predicting a specific biological replicate could increase performance (e.g. if the model saw the replicate during training), but would be misleading because it would not generalize to the samples we actually care about. Thirdly, we aimed to make this framework sustainable in the long term by distributing responsibilities to those making models and benchmark datasets, and having minimal centralized maintenance. New models/datasets need only to implement the API to seamlessly communicate with all others that already exist in the framework. While this adds some initial overhead, the existing models and benchmarks in GAME will mean that implementing the GAME API will become the most efficient way to complete evaluations.

In GAME, models are encapsulated in Predictors and benchmarks are encapsulated in Evaluators, which communicate via the API protocol. The Evaluator module is a software client that is responsible for evaluating models on a specific benchmark, while the Predictor module is a server that listens for incoming requests and responds with the model's predictions for the requested tasks. We also have included a Matcher module which uses standardized queries to a local LLM<sup>17</sup>. Matcher is responsible for mapping requested tasks with tasks the model knows how to do (e.g. species, cell types, and molecules). For example, the Matcher must map the cell types requested by the Evaluator to the closest matched cell types that the Predictor is able to predict. The Matcher is implemented as another server for which the Predictor serves as the client. A GAME evaluation occurs in 7 steps (**Fig. 1**). (1) The Evaluator requests predictions from a Predictor. (2) The Predictor parses the request and, if needed, consults the Matcher to determine how it can best serve up the tasks requested. (3) The Matcher finds the mapping between the requested tasks and tasks the predictor can do. (4) The Matcher sends this information back to the Predictor. (5) The Predictor processes the request, by sending it to the model and reformatting the model's predictions in the required standard format. (6) The predictions are returned to the Evaluator. (7) Finally, the Evaluator calculates the model's performance. The Evaluator builders are responsible for determining what information the Predictors require for their predictions and what the best evaluation metrics are for their benchmarks. The Predictor builders are responsible for taking a request for predictions on a set of DNA sequences and determining how it can best fulfill the request, given what the model can predict (or decline to predict if it cannot do so). The Evaluators, Predictors, and Matcher are containerized using Apptainer to facilitate sustainable model evaluation across computing environments, improving our ability to compare models<sup>18</sup>. Inside the containers, the modules can be written in any language. However, to facilitate the adoption of this approach, we provide examples and templates for functions implemented in Python with seamless GPU support that users can modify to fit their needs.

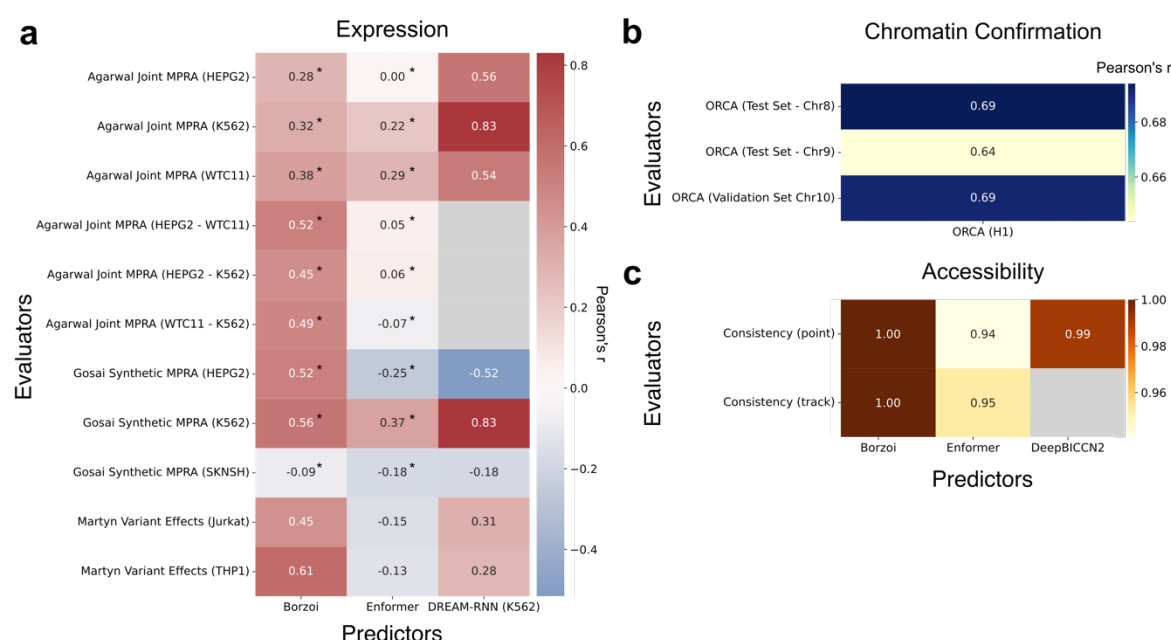
Including a separate Matcher module provides several advantages. The cell types requested by the Evaluator are likely imperfect matches to the cell types the Predictor can predict (e.g. cardiomyocyte vs. heart or HEK293FT vs. HEK293). Similarly, the Predictor may not include the requested species and so would need to know which of the species it includes is closest. The Matcher resolves such discrepancies by automating the alignment of tasks. This ensures that performance measures are not influenced by the Predictor builder's understanding of cellular, molecular, or species relationships. Further, as our understanding of species and cell types evolves, the Matcher module can be upgraded while remaining compatible with existing GAME modules.



**Figure 1: GAME framework.** GAME includes three modules: The Evaluator, containing a benchmark dataset; the Predictor, encompassing a sequence-to-activity model; and the Matcher, capturing relationships between tasks. All GAME modules are inherently interoperable by communicating in the GAME API protocol over TCP. For each benchmark, the Evaluator requests a prediction from the Predictor, which consults the Matcher to determine the closest task the Predictor can complete. Once the Matcher returns the best match, the Predictor will complete its prediction and return it to the Evaluator, which will evaluate performance. Members of the genomics community will contribute modules to enable continual evaluation of more models across more benchmarks.

We applied GAME to several models and two representative benchmarking tasks – gene expression and chromatin confirmation (**Fig. 2**). For gene expression, we implemented three Evaluators, two using MPRA datasets (both genomic and synthetic)<sup>19,20</sup>, and one of measured effects of synthetic variants<sup>21</sup>, each of which can output distinct evaluation metrics. For example, the Agarwal et al. MPRA Evaluator evaluates models on their ability to predict gene expression separately for each cell type, and also on how well models predict cell type-specific differences in expression (**Fig. 2a**)<sup>19</sup>. The Gosai et al. MPRA Evaluator data includes synthetic (non-genomics) MPRA probes, and so is an ideal benchmark for genome-trained models. Similarly, the Martyn data represents the expression effects of synthetic genetic variation introduced by genome editing, and so is a good task benchmarking models on variant effects with minimal risk of train-test leakage. We created Predictors for Borzoi, Enformer, and DREAM-RNN, and evaluated each on each expression Evaluator (**Fig. 2a**). Borzoi and Enformer are both large sequence window genome-trained state-of-the-art sequence-to-function models and have been used in numerous benchmarking papers<sup>22,23</sup>. The DREAM-RNN Predictor is a state-of-the-art short-sequence sequence-to-function model<sup>24</sup>, in this case trained on Agarwal K562 MPRA data<sup>19</sup>. As a more complex example we also created a 3D chromatin confirmation (**Fig. 2b**) Evaluator and Predictor based on

Orca<sup>25</sup>. Finally, we developed Consistency Evaluators that can be used to assess models that predict DNA accessibility, confirming correlated predictions between forward and reverse complement sequences (**Fig 3b**) for Borzoi, Enformer, and the recently-described ATAC-seq model DeepBICCN2<sup>26</sup>



**Figure 2: Sample benchmarking done with GAME.** **a**, Expression evaluation tasks. Models (x axis) were evaluated for their correlation to measured expression levels (colours) across a variety of tasks (y axis). **b**, Chromatin conformation tasks. Correlation of Orca predictions vs measured chromatin contact frequencies (colours) for two Orca test-set chromosomes and one validation-set chromosome (y axis). **c**, Consistency evaluation for accessibility. Point and track based accessibility consistency evaluators (y axis) were used to evaluate the correlation between predictions for forward and reverse complement sequences (colours) in models of DNA accessibility (x axis). \*Correlation values from down sampled datasets. Grey cells mark values that were not calculated due models that could not complete the Evaluator's requested task.

GAME offers many advantages that will streamline benchmarking and model use more generally. The common protocol enables any model to communicate with any benchmark (although not all combinations may make sense), ensuring that when one creates a new GAME module, it is immediately compatible with all the existing modules. Existing Predictors and Evaluators remain frozen so that future benchmarks done with them are identical; future evaluations of models on these benchmarks will be directly comparable to the results shown in **Fig. 2**. The anonymity of API requests means that all models will be evaluated equally. The modules communicate via TCP/IP sockets and so can be run locally or remotely, potentially preserving privacy, for example, in cases where models have proprietary weights. Because GAME is decentralized and community-driven, it will be much more scalable and easily maintained. Contributors can easily add their own benchmarks and models, and can maintain them as long as they remain relevant. We have already implemented 5 models and 6 benchmarks in GAME.

As GAME gains momentum, the fastest and easiest way of performing a new benchmark will be to implement GAME, further incentivizing this community effort. We anticipate that GAME will drive genomics progress in several ways. GAME allows one to easily compare all models across all benchmarks, which will reveal model limitations and low-quality

benchmarks. For instance, a model that was overfitted to a current benchmark would be revealed as it fails to generalize to the benchmarks that arise after its release. Such a framework also facilitates continual evaluation of state-of-the-art models on a diverse set of tasks, enabling researchers to quickly identify the models that are most well-suited to their specific tasks. For instance, if one is interested in predicting the effects of genetic variants in heart, one can find the closest benchmark to determine which model is best for the task. To showcase GAME in a reasonable time frame with reduced computational costs, we down-sampled certain Evaluator datasets (**Fig. 2**). However, Evaluators and Predictors can split up the dataset, enabling parallelization and better utilization of available resources. While we anticipate that Matcher will be useful outside of GAME, determining the best way to match tasks turned out to be a substantial challenge. Nevertheless, the Matcher can be continually improved due to its modular nature. Finally, having all models share a common language means that other software can use GAME to facilitate model reuse and swapping between models more easily. Its modular design will keep GAME in play, evolving alongside the ever-changing field of genomics.

## Methods

### Information passing

The client and server code of an API can be written in any programming language<sup>27</sup>. However, due to the prevalence of Python programming in genomics and model development<sup>28,29</sup>, the GAME API and corresponding error-checking functions are written in Python. Python sockets can connect servers to clients to communicate with each other over local or remote networks<sup>30</sup>. Python sockets use Transmission Control Protocol (TCP), which provides benefits such as detecting lost information when sending information over a network and controlling the order of data when transmitted<sup>31</sup>.

The Predictor server will bind to a user-specified “HOST,” an IP address or hostname, and “PORT,” a TCP port. Once the server binds, it will wait and listen for incoming connections. The Evaluator client must connect to the same HOST and PORT pair to send a request to the server. The data that is transmitted via the TCP socket must be converted to bytes before it is sent. The Evaluator and Predictor can both be on a local computer, HPC platform, or even in remote configurations, ensuring flexibility in how they communicate. This capability is also beneficial for Predictor developers working with proprietary or closed-source models, allowing them to integrate their work into the GAME framework. Those models can be embedded into the Predictor codebase and hosted as a server on their own network, maintaining full control and privacy over their model weights.

### Communication protocol

In GAME, the communication protocol distinguishes between the initial format of user-provided data and the standardized formats used for data streaming between GAME modules. While users can store benchmark data in various initial file formats (e.g. .json, .npy, .txt, .xlsx), these inputs must first be converted to the correct format before transmission. Most typically, this is implemented as a helper function that converts the input file into JavaScript Object Notation (JSON)-like dictionary structure. It is essential that this structure adheres to the predefined JSON schema of GAME to ensure data compatibility and

proper interpretation by subsequent modules. Data in the JSON object is stored as key: value pairs<sup>32</sup>.

Once the data is in this standardized dictionary representation, it is ready for transmission. By default, GAME utilizes JSON for this inter-module data exchange, due to its popularity, speed in generation and parsing, and support via Python's standard library<sup>33</sup>. We also allow information to be transmitted using MessagePack (MsgPack), a binary serialization format, which helps increase speed when returning large arrays of predictions<sup>34,35</sup>, provided both the Evaluator and Predictor modules are configured to handle it. The Evaluator, therefore, serializes the prepared dictionary structure into either a JSON or a MsgPack stream before sending the required information to the Predictor. If the task requested is not an exact match to one of the Predictor's model outputs, the Matcher module can be utilized to align the requested cell types, species, and molecules with those supported by the Predictor. Once the prediction is complete, the predictions are reformatted to the negotiated return format and returned to the Evaluator. JSON and MSGPACK files are easily loaded into Python and accessed as dictionaries, making them straightforward to work with<sup>36</sup>. For implementation guidance, sample files and code can be found on GitHub.

## Module responsibilities

The Evaluator and Predictor modules are designed to have specific responsibilities that follow the core principles of the API, such as minimizing information sent to a Predictor, allowing dataset creators the power to evaluate the tasks they are interested in, and allowing model builders the freedom to parse sequences to comply with model specifications as they see fit. The main responsibilities of Evaluators are to choose what predictions they would like to obtain from the Predictor and what information the Predictor needs to fulfil that request, including: whether the Predictor should return points (a single value) or tracks (values across a sequence), whether there are constant flanking sequences that could provide relevant context (e.g. the vector backbone of an MPRA vector), where in the sequence window the predictions should be provided (e.g. the location of a promoter), how to use the expression predictions based on sense or antisense strand. This information is encoded in the request that is sent to the Predictor. Predictors, conversely, are responsible for parsing the sequences to meet model input size requirements (pad, trim, or add other flanks to the sequences), aggregate across tracks (e.g. replicates of an experiment) or bins (e.g. if the prediction window overlaps multiple bins), deciding which cell type/species/molecule to use (with help from the Matcher, if desired), and returning the predictions in the standard format. Once the predictions are returned, it is the Evaluator's responsibility to calculate the appropriate metrics for each benchmarking task and output these results to a file, using a standard reporting format. Additional details of the API specification and examples are available on GitHub.

## Containerization

Containers provide a secure and robust way to run code across any platform, removing troublesome dependency issues. Since most ML models are trained and evaluated on high-performance computing (HPC) platforms, we used Apptainer for GAME module containers. Unlike Docker, Apptainer is specifically designed for HPC environments and does not require root access, a critical advantage since most HPC systems restrict root privileges for security reasons<sup>18</sup>. By containerizing each Evaluator and Predictor, the code within each

container is isolated, enabling each to use its own version of any language without issue. Additionally, in contrast to Docker, Apptainer containers have no network isolation, meaning that they can communicate with each other using IP addresses and ports without having to create a network for them to communicate over.

## Maintenance and Community Contributions

The long-term maintenance of this framework is designed to be simple and scalable. We have made community-maintainable lists of GAME modules available on GitHub, which then needs only to be updated with the locations of new modules. For instance, someone could create a new model, package it into a Predictor, and then issue a pull request to update the list of Predictors with their own. Subsequently, users can pull the latest lists of Predictors and test them on the latest list of Evaluators. The module list includes their names, a description of their function, and links to the locations where containers are stored. Zenodo provides an ideal location for housing Evaluator and Predictor containers because it is durable and has the additional benefit of scanning uploads for viruses to ensure the safety of containers for users to pull<sup>37,38</sup>.

## Automated and externalized cell type, molecule, and species matching

GAME includes a module called “Matcher” that automatically maps the Evaluator’s requested species, measured molecule, and cell type with what a Predictor can provide. The Matcher is designed to perform this task by interpreting the relationship between terms through lexical, syntactic, and semantic matching. Lexical matching handles cases of direct string correspondence, such as finding the exact token ‘A549’ within a more descriptive choice like ‘lung adenocarcinoma cell line: A549’. Syntactic matching addresses structural variations and common abbreviations, such as ‘hek-293’ or ‘SKNSH’ to ‘HEK293’ or ‘SK-N-SH’, respectively. Semantic matching uses biological knowledge to connect different terms that refer to the same entity, such as mapping the description ‘chronic myelogenous leukemia cell line’ to a canonical cell line name, ‘K562’.

The matcher uses a local large language model – Gemma 3, which is a collection of very lightweight and efficient models that can run effectively on a single GPU or TPU<sup>17</sup>. Running Gemma 3 locally means that the model operates directly on the hardware of the system it is deployed in, ensuring data privacy, reducing reliance on external cloud APIs, and lowering operational costs, which are crucial for sensitive or high-throughput workflows.

The Matcher module utilizes Ollama to run and manage Gemma 3 locally<sup>39</sup>. Ollama is an open-source platform that simplifies hosting large-language models (LLMs) on local hardware by packaging models and providing tools for their management. Ollama helps abstract away many underlying complexities of model backends using inference engines and can automatically leverage available GPU resources for accelerated performance. This ease of setup and serving is key to the containerized and reproducible design of the Matcher module<sup>40,41</sup>. Additionally, the availability of an official Docker image for Ollama further streamlines the creation of portable and reproducible runtime environments. When a model is run, Ollama exposes a local API endpoint (on port 11434). This allows applications, like the Predictors in GAME, to send requests to the LLM and receive responses.

To enable the Matcher module Python environment to interact with the Gemma 3 model, the LangChain framework is utilized<sup>42</sup>. LangChain is an open-source library designed for

developing applications that incorporate LLMs<sup>42,43</sup>. It provides a collection of tools that assist in managing prompts, interacting with LLM APIs (including local instances like those served by Ollama), and constructing more extensive LLM driven pipelines. Within the Matcher module, the `langchain-ollama` package provides specific client classes, like `OllamaLLM`, which facilitate communications with the local Ollama API. This integration enables the Matcher to send structured input queries – comprising the fuzzy input term (Evaluator request) and the list of choices (what the Predictor can provide predictions on) – to the Gemma 3 model through the Ollama server and then to receive the best identified match. To create structured queries consistently, the `ChatPromptTemplate` class is employed from LangChain, which supports the goal of reproducible matching. By automating this process, we reduce potential biases due to memorization of cell-type relationships. This also circumvents the process of manual selection or literature review to select closest cell types. Matcher can also be used to align species or tasks (e.g. ‘NF-kB’ to ‘NFKB1’).

A key challenge in this process is that a Predictor may have hundreds of available choices, a list too long to fit within the context window of the LLM<sup>44,45</sup>. To overcome this, the Matcher implements a chunking strategy. The complete list of choices is first divided into smaller, manageable chunks. The LLM then performs a ‘tournament-style’ elimination, finding the best possible match, the ‘champion’, within each individual chunk. Finally, a championship round is held where the LLM evaluates only the list of champions to determine the single best overall match. This method makes the matching process scalable to arbitrarily long lists of choices.

## Data and Code availability

API specifications, code and additional documentation can be found on GitHub (<https://github.com/de-Boer-Lab/Genomic-API-for-Model-Evaluation/tree/main>). List of current Modules can be found here: [https://github.com/de-Boer-Lab/GAME\\_modules](https://github.com/de-Boer-Lab/GAME_modules)

## Acknowledgements

We would like to thank J. Gagneur, T. Mauermeier, J. Shendure, C. Qiu, D. Calderon, A. Gao, P. Fradkin, B.J Frey, G. Eraslan, L. Gunsalus, S. Nair, D. Kelley, R. Das, S. Kundu, I. Raine, V. Hecht, A. Kundaje, Z. Avsec, J. Engreitz, A. Gschwind, M. Montgomery, M. Weilert, C. McAnany for helpful feedback and discussion during the API design and implementation process. We would also like to thank M. Krzywinski for design help with Figure 1. M.A.W was supported by National Institutes of Health (USA) R01GM121755. J.Z, S.D and A.F were supported by Additional Ventures, Keck Foundation, Biswas Family Foundation, Gladstone Institutes. J.Z was supported by Cancer Prevention and Research Institute of Texas grant RR190071 and National Institutes of Health (NIH) grants DP2GM146336.

## Author Contributions

I.L and C.G.D conceived and designed GAME. I.L and S.P set up GAME code base and documentation. R.G set up codebase for Orca Evaluator and Predictor. L.M and N.K contributed DeepBICCN2 Predictor. M.M provided data for variant effects Evaluator. I.L, S.P, and C.G.D wrote the paper. K.P, H.C, A.K, J.Z, L.P, M.A.W, A.L, I.K, I.K, D.P, and K.P provided feedback on the manuscript. D.D and W.H provided feedback for Matcher and

helped create early implementations of the Matcher module. All other authors were involved in either the design process for the API specifications, feedback and/or testing API tutorials and setup.

## Competing Interests

V.A. is an employee of Sanofi. A.L. is an employee of Genentech, inc. The remaining authors declare no competing interests.

## References

1. Guerrini, M. M., Oguchi, A., Suzuki, A. & Murakawa, Y. Cap analysis of gene expression (CAGE) and noncoding regulatory elements. *Semin Immunopathol* **44**, 127–136 (2022).
2. Tang, Z., Toneyan, S. & Koo, P. K. Current approaches to genomic deep learning struggle to fully capture human genetic variation. *Nat Genet* **55**, 2021–2022 (2023).
3. Sasse, A. *et al.* Benchmarking of deep neural networks for predicting personal gene expression from DNA sequence highlights shortcomings. *Nat Genet* **55**, 2060–2064 (2023).
4. Huang, C. *et al.* Personal transcriptome variation is poorly explained by current genomic deep learning models. *Nat Genet* **55**, 2056–2059 (2023).
5. Robson, E. S. & Ioannidis, N. M. GUANinE v1.0: Benchmark Datasets for Genomic AI Sequence-to-Function Models. *bioRxiv* 2023.10.12.562113 (2024). doi:10.1101/2023.10.12.562113.
6. Karollus, A., Mauermeier, T. & Gagneur, J. Current sequence-based models capture gene expression determinants in promoters but mostly ignore distal enhancers. *Genome Biology* **24**, 56 (2023).
7. Kryzhtalovich, A., Schwede, T., Topf, M., Fidelis, K. & Moult, J. Critical Assessment of Methods of Protein Structure Prediction (CASP) – Round XIII. *Proteins* **87**, 1011–1020 (2019).
8. Krizhevsky, A. & Hinton, Geoffrey. Learning Multiple Layers of Features from Tiny Images. (2009).
9. Grešová, K., Martinek, V., Čechák, D., Šimeček, P. & Alexiou, P. Genomic benchmarks: a collection of datasets for genomic sequence classification. *BMC Genom Data* **24**, 25 (2023).
10. Stolovitzky, G., Prill, R. J. & Califano, A. Lessons from the DREAM2 Challenges. *Annals of the New York Academy of Sciences* **1158**, 159–195 (2009).
11. Stolovitzky, G., Monroe, D. & Califano, A. Dialogue on Reverse-Engineering Assessment and Methods. *Annals of the New York Academy of Sciences* **1115**, 1–22 (2007).
12. Kaggle Competitions. <https://www.kaggle.com/competitions>.
13. Bloch, J. How to design a good API and why it matters. in *Companion to the 21st ACM SIGPLAN symposium on Object-oriented programming systems, languages, and applications* 506–507 (Association for Computing Machinery, New York, NY, USA, 2006). doi:10.1145/1176617.1176622.
14. Lamothe, M., Guéhéneuc, Y.-G. & Shang, W. A Systematic Review of API Evolution Literature. *ACM Comput. Surv.* **54**, 171:1–171:36 (2021).
15. Avsec, Ž. *et al.* The Kipoi repository accelerates community exchange and reuse of predictive models for genomics. *Nat Biotechnol* **37**, 592–600 (2019).

16. Stergachis, A. B., Debo, B. M., Haugen, E., Churchman, L. S. & Stamatoyannopoulos, J. A. Single-molecule regulatory architectures captured by chromatin fiber sequencing. *Science* **368**, 1449–1454 (2020).
17. Team, G. *et al.* Gemma 3 Technical Report. Preprint at <https://doi.org/10.48550/arXiv.2503.19786> (2025).
18. Pasupuleti, R., Vadapalli, R., Mader, C. & Milenkovic, V. J. Apptainer-Based Containers for Legacy and Platform Dependent Machine Learning Applications on HPC Systems. in *2024 IEEE International Conference on Big Data (BigData)* 6083–6091 (2024). doi:10.1109/BigData62323.2024.10825376.
19. Agarwal, V. *et al.* Massively parallel characterization of transcriptional regulatory elements. *Nature* **639**, 411–420 (2025).
20. Gosai, S. J. *et al.* Machine-guided design of cell-type-targeting cis-regulatory elements. *Nature* **634**, 1211–1220 (2024).
21. Martyn, G. E. *et al.* Rewriting regulatory DNA to dissect and reprogram gene expression. *Cell* **188**, 3349–3366.e23 (2025).
22. Avsec, Ž. *et al.* Effective gene expression prediction from sequence by integrating long-range interactions. *Nat Methods* **18**, 1196–1203 (2021).
23. Linder, J., Srivastava, D., Yuan, H., Agarwal, V. & Kelley, D. R. Predicting RNA-seq coverage from DNA sequence as a unifying model of gene regulation. 2023.08.30.555582 Preprint at <https://doi.org/10.1101/2023.08.30.555582> (2023).
24. Rafi, A. M. *et al.* A community effort to optimize sequence-based deep learning models of gene regulation. *Nat Biotechnol* 1–11 (2024) doi:10.1038/s41587-024-02414-w.
25. Zhou, J. Sequence-based modeling of three-dimensional genome architecture from kilobase to chromosome scale. *Nat Genet* **54**, 725–734 (2022).
26. Kempynck, N. *et al.* CREsted: modeling genomic and synthetic cell type-specific enhancers across tissues and species. 2025.04.02.646812 Preprint at <https://doi.org/10.1101/2025.04.02.646812> (2025).
27. Sinha, A. Client-server computing. *Commun. ACM* **35**, 77–98 (1992).
28. Zou, J. *et al.* A primer on deep learning in genomics. *Nat Genet* **51**, 12–18 (2019).
29. Cock, P. J. A. *et al.* Biopython: freely available Python tools for computational molecular biology and bioinformatics. *Bioinformatics* **25**, 1422–1423 (2009).
30. Hassan, G. M., Hussien, N. M. & Mohialden, Y. M. Python TCP/IP libraries: A Review. *International Journal Paper Advance and Scientific Review* **4**, 10–15 (2023).
31. Xue, M. & Zhu, C. The Socket Programming and Software Design for Communication Based on Client/Server. in *2009 Pacific-Asia Conference on Circuits, Communications and Systems* 775–777 (2009). doi:10.1109/PACCS.2009.89.
32. Lv, T., Yan, P. & He, W. On Massive JSON Data Model and Schema. *J. Phys.: Conf. Ser.* **1302**, 022031 (2019).
33. Pezoa, F., Reutter, J. L., Suarez, F., Ugarte, M. & Vrgoč, D. Foundations of JSON Schema. in *Proceedings of the 25th International Conference on World Wide Web* 263–273 (International World Wide Web Conferences Steering Committee, Republic and Canton of Geneva, CHE, 2016). doi:10.1145/2872427.2883029.
34. MessagePack: It's like JSON. but fast and small. <https://msgpack.org/>.
35. Viotti, J. C. & Kinderkheadia, M. A Survey of JSON-compatible Binary Serialization Specifications. Preprint at <https://doi.org/10.48550/arXiv.2201.02089> (2022).
36. Python JSON: Read, Write, Parse JSON (With Examples). <https://www.programiz.com/python-programming/json>.

37. Sicilia, M.-A., García-Barriocanal, E. & Sánchez-Alonso, S. Community Curation in Open Dataset Repositories: Insights from Zenodo. *Procedia Computer Science* **106**, 54–60 (2017).
38. Peters, I., Kraker, P., Lex, E., Gumpenberger, C. & Gorraiz, J. I. Zenodo in the Spotlight of Traditional and New Metrics. *Front. Res. Metr. Anal.* **2**, (2017).
39. Ollama. <https://ollama.com>.
40. ollama/ollama. Ollama (2025).
41. Marcondes, F. S. *et al.* Using Ollama. in *Natural Language Analytics with Generative Large-Language Models: A Practical Approach with Ollama and Open-Source LLMs* (eds. Marcondes, F. S. *et al.*) 23–35 (Springer Nature Switzerland, Cham, 2025). doi:10.1007/978-3-031-76631-2\_3.
42. Chase, H. LangChain. (2022).
43. Topsakal, O. & Akinci, T. C. Creating Large Language Model Applications Utilizing LangChain: A Primer on Developing LLM Apps Fast. *ICAENS* **1**, 1050–1056 (2023).
44. Liu, N. F. *et al.* Lost in the Middle: How Language Models Use Long Contexts. Preprint at <https://doi.org/10.48550/arXiv.2307.03172> (2023).
45. What is a context window? | IBM. <https://www.ibm.com/think/topics/context-window> (2024).