Fine-Mapping Tools: an interactive framework for dissecting disease-associated genetic loci with functional genomics data

by

Peter HT Nguyen

Submitted to the Department of Electrical Engineering and Computer Science
in partial fulfillment of the requirements for the degree of

Masters of Engineering in Computer Science and Molecular Biology

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

June 2017

Author .....
Department of Electrical Engineering and Computer Science
May 26, 2017

Certified by .....
Manolis Kellis
Professor
Thesis Supervisor

Accepted by .....
Christopher Terman
Chairman, Masters of Engineering Thesis Committee

Fine-Mapping Tools: an interactive framework for dissecting disease-associated genetic loci with functional genomics data

by

Peter HT Nguyen

Submitted to the Department of Electrical Engineering and Computer Science
on May 26, 2017, in partial fulfillment of the
requirements for the degree of
Masters of Engineering in Computer Science and Molecular Biology

Abstract

Fine mapping causal SNPs from GWAS summary statistics is hard. Although many frameworks exist to support fine mapping, some of which leverage epigenomic contexts to increase predictive power, they fail to provide interactivity. Here, we introduce Fine-Mapping Tools (fm-tools), a framework for doing interactive and iterative fine mapping. Fm-tools provides a harmonized data store and implements a number of algorithms for fine mapping — one of which is the custom RiVIERA-mini, an efficient Bayesian inference framework — and exposes them via a rich API that can be plugged into a variety of services (e.g., web applications for visualization). Most importantly, fm-tools allows scientists to interactively and iteratively explore dynamically generated hypotheses, as demonstrated by a case study for celiac disease. In summary, fm-tools standardizes the way fine mapping is done, reduces the overhead of fine mapping for scientists and of algorithm development for researchers, and paves the way towards achieving real-time personalized medicine.

Thesis Supervisor: Manolis Kellis
Title: Professor

Acknowledgments

I am thankful to Manolis Kellis for having me in Kellis Lab, and for providing invaluable advice and mentorship throughout my graduate tenure; to Yue Li for having intellectual meetings and curating the data sets for fm-tools; and to Yongjin Park, Lei Hou, Alvin Shi, and other members of the Kellis Lab for having intellectual meetings.

I am thankful to my long-time friend and colleague Tiange Zhan for contributing to the implementation of RiVIERA-mini, participating in the celiac disease case study, and providing constant love and support.

Lastly, I am thankful to my parents Long Nguyen and Ha Do for raising me, giving me an opportunity to attend MIT, and providing constant love and support.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Although genome-wide association studies (GWAS) have enabled the identification of genetic variants that are associated with a given phenotype, identifying the causal variants (fine mapping) to discover disease pathogenesis remains a central problem in human genetics and personalized medicine. It is well known that fine mapping the causal variants from GWAS summary statistics is hard because linkage disequilibrium [1] can cause non-causal variants to be more statistically associated with the phenotype than they should be. In addition, discovering disease pathogenesis from a given set of GWAS-implicated variants is hard because a majority of those variants map to non-coding regions of the genome, and it is hard to infer the functional roles of such variants in disease pathogenesis [2].

To facilitate the fine mapping of non-coding causal variants, researchers have developed tools that allow scientists to analyze GWAS variants in the context of various epigenomic annotations. Epigenomic annotations identify regions in the genome that are subject to certain regulatory functions [3]. For example, certain regions of the genome are promoters, while others are enhancers that are only active in a certain tissue type (e.g., immune). Analyzing a variant within its epigenomic context can confer functional information about the variant, thereby allowing researchers to infer the functional roles of non-coding variants and fine map causal variants. One tool that allows scientists to analyze variants within epigenomic contexts is HaploReg [4]. HaploReg is an interactive web tool that allows end

users to mine for epigenomic annotations that pertain to a given set of variants, often in the context of a disease phenotype.

The problem with a tool like HaploReg is that it only presents static data. This implies that scientists cannot make use of new data sets, whether they have new GWAS data sets to analyze or they want to analyze existing data sets in the context of newly identified epigenomic annotations. More importantly, though, static tools like HaploReg do not allow scientists to interactively condition their analyses on dynamically generated hypotheses. For example, suppose that a scientist has used HaploReg to identify causal variants for celiac disease in the context of all known epigenomic annotations. If the scientist subsequently decides to condition the analysis on only the epigenomic annotations that are relevant to the immune system, then we expect the identity of the causal variants to differ. Unfortunately, tools like HaploReg do not support doing such interactive analyses. In addition, HaploReg does not re-calculate the GWAS probabilities of association based on newly selected epigenomic contexts, because it is only responsible for presenting static data.

To allow scientists to infer causal variants in the context of different epigenomic contexts, researchers have proposed a number of machine learning algorithms that jointly model the GWAS summary statistics (in particular, the GWAS p-values of association) and the epigenomic contexts. For example, fgwas [5] is a statistical model that leverages the epigenomic context to jointly infer causal variants and causal epigenomic annotations for a phenotype. The advantage here is that scientists can use the causal epigenomic annotations as contexts for identifying novel genomic loci that are more likely to contain causal variants for a phenotype. Similarly, GPA [6] is an expectation-maximization algorithm that allows scientists to specify a set of epigenomic annotations as context for fine mapping causal variants. Unfortunately, tools like fgwas and GPA are limited because they only allow scientists to do inference with a small number of epigenomic annotations at a time, because of computational constraints. In addition, they only allow scientists to analyze a single phenotype at time, which prevents them from leveraging the functional correlation between phenotypes (at the epigenomic level) to better infer causal variants.

One algorithm designed to address the above problems is RiVIERA [7]. RiVIERA is a

framework that, like fgwas and GPA, jointly models the GWAS summary statistics and the epigenomic contexts, and that aims to infer causal variants and causal epigenomic annotations across one or more related phenotypes. The main advantage of RiVIERA is that it provides a robust and efficient Bayesian framework for doing inference. This allows scientists to fine map causal variants in the context of hundreds of genome-wide and tissue-specific epigenomic annotations. In addition, RiVIERA allows scientists to leverage the shared regulatory mechanisms between similar phenotypes to increase the power in identifying causal variants and causal epigenomic annotations.

Even though a robust Bayesian inference framework like RiVIERA has many advantages, it still falls short in providing scientists with a tool that facilitates interactive and iterative fine mapping analyses. First, the RiVIERA tool has many dependencies, so it is hard for scientists to immediately download the tool and run it in any generic computing environment. Second, even if the scientist manages to download the tool in a suitable environment, many manual operations must still be executed to get the data sets in a format that can be specified as input to the RiVIERA tool. These manual operations include downloading the data sets from various disparate sources and converting them into the appropriate formats. Third, even if the scientist manages to get the appropriate inputs and runs the tool, the RiVIERA tool is largely designed for batch processing, which is not amenable to interactivity because of large latencies and lack of support for end user interruptions. This means that, like HaploReg, RiVIERA is not suited to allowing scientists to dynamically condition interactive analyses. Fourth, RiVIERA is not designed for client-server architectures deployed on the internet, especially in light of unreliable network connectivity, so it is difficult to deploy intricate web applications for interactivity and visualization.

In this paper, we introduce Fine Mapping Tools (fm-tools), a framework for doing interactive and iterative fine mapping analyses. Fm-tools provides a harmonized data store that supports fine mapping analyses by aggregating genetic and epigenomic data sets from disparate sources into a consistent schema. In addition, fm-tools supports the creation of new data sets that can be easily integrated with existing data sets, at the discretion of the end user. Fm-tools also implements an inference engine (based on the RiVIERA algorithm) for doing

fine mapping in configurable epigenomic contexts. Most importantly, fm-tools exposes a rich and extensible API that supports interactivity and iteration. This implies that scientists can interactively explore a number of hypotheses and dynamically adjust analyses on the fly. Fm-tools solves the problems inherent in static tools like HaploReg and in batch processing tools like RiVIERA by bringing the data sets and the inference engine together, and exposing them via a modular API.

In deploying fm-tools, we specifically aim to target two audiences. First, we aim to cater to scientists who want to do fine mapping analyses. These may be scientists who do the GWAS experiments and who want to further analyze the GWAS outputs in the context of various tissue types. Alternatively, these may be scientists who are developing drugs for a specific disease phenotype, and who are searching for the best SNPs and/or histone modifications to use as targets for the drug. To further this end, fm-tools exposes an API that is both interactive and easy to use for scientists in the field, and abstracts away the data curation and tools necessary for fine mapping.

Second, we aim to provide a platform for research and development of new machine learning algorithms that support fine mapping. The key development of fm-tools that facilitates this is the harmonized data store. One of the challenges in developing and testing new algorithms is curating the necessary data sets. Fm-tools naturally provides this capability. In addition, the fm-tools API is general enough to support a variety of algorithms in the inference engine, while still providing the same interactivity, consistency, availability, and performance guarantees to the client.

The remainder of this paper is organized as follows. Section 2 highlights several of the design decisions behind the implementation of fm-tools. Section 3 discusses the system architecture. Section 4 and Section 5 discuss the data model and the relevant implementation experience. Section 6 describes the API. Section 7 discusses the existing RiVIERA algorithm and highlights the modified algorithm that is deployed in fm-tools. Section 8 illustrates a case study of a scientist using fm-tools for interactive and iterative fine mapping of celiac disease. Section 9 acknowledges the limitations of fm-tools and mentions some future work.

Chapter 2

Design decisions

We discuss the design decisions behind the implementation of fm-tools, in light of several system properties.

2.1 Client-server architecture

One of the key properties of fm-tools is that it is based on a client-server architecture. We believe that this architecture best allows fm-tools to achieve its goal of providing a platform for doing interactive and iterative fine mapping analyses. In particular, the server is able to address one of the main problems in using RiVIERA for fine mapping: many manual operations must be executed to curate the data sets before a scientist is able to run RiVIERA against them. For example, the server implements a harmonized data store that hosts the data sets for scientists to immediately use; no manual operations must be executed. In addition, the architecture allows us to centralize the work involved in curating the data sets to the server component (discussed in more detail in Section 3.1). This means that data curation only needs to be done once for a given data set; once the data set is hosted, any scientist can use it for fine mapping.

Another advantage of the client-server architecture is that it abstracts away many dependency and computing requirements from the end user. The end user no longer needs to

worry about which tools to download and which computing environment is necessary. All the end user needs is a network connection and a desire to do fine mapping.

2.2 Modularity, resumability, and idempotence

Because fm-tools is based on a client-server architecture, it is prone to the problems inherent in such an architecture. One of these problems is network connectivity: it is not always possible for clients to communicate with the server. In light of this, fm-tools is designed with modularity, resumability, and idempotence in mind. These properties are especially important in fine mapping because it is a long process. If the network connection dies in the middle of a pipeline, then it is important that the pipeline can be resumed as close to the point of failure as possible, thereby saving time, money, and headaches.

A number of things must be done to achieve resumability in fine mapping without compromising it overall. Intuitively, the inference engine must be broken down into modular components. However, the API must also expose these components in a way that minimizes the amount of data that needs to be transferred to do a complete fine mapping operation; this is important for ensuring that the overhead of breaking the inference engine down into components is low. More importantly, the system must provide some form of idempotence: repeated operations must not yield different outputs. For example, a repeated operation against the inference engine due to network connectivity problems should output the same inference estimates as the first operation. Section 6.2 discusses these properties in more detail.

2.3 Interactivity

Interactivity is one of the main goals of fm-tools. We aim to allow scientists to interactively explore a number of hypotheses when doing fine mapping: for example, a scientist should be able to do fine mapping in a global epigenomic context, and then follow-up with fine

mapping in a tissue-specific context, and so forth. To achieve interactivity, the inference engine must be able to output results quickly; that is, it must have high performance. Because fine mapping is a fairly heavyweight operation, it is hard to support interactivity and performance without compromising on accuracy. A batch processing tool like RiVIERA achieves accuracy at the cost of interactivity and performance. Internally, RiVIERA must iterate through hundreds of steps of inference (discussed more in Section 7.1), resulting in latencies on the order of minutes for most production pipelines. This amount of latency is not acceptable for fm-tools, so we design the system with the goal of optimizing for interactivity and performance, at the cost of accuracy (see the description of RiVIERA-mini in Section 7.2). This is generally acceptable because the API does support doing interactive analyses for a longer period of time to yield better accuracies at the discretion of the end user.

2.4 Scalability

Scalability is important for fm-tools because it is responsible for big data, in terms of both data volume and compute requirements. As such, it is important that fm-tools can scale linearly with the size of the data, as well as with the number of end users. Section 3.1 discusses in more detail how the architecture supports scalability at the application level.

2.5 Consistency and availability

Although consistency and availability are important properties of any production system [8], we choose to not focus on them in this paper. We believe that this is okay because the system is currently in alpha mode, so it is not under heavy usage. In addition, we do have some primitive mechanisms in place to address these properties.

Indeed, fm-tools does make some consistency guarantees, albeit somewhat coarse. In the steady state (in which no new data sets are being created by API clients), we ensure that all operations are linearizable [9]. In the dynamic state, we ensure that fine mapping operations

(discussed in Section 6), can only operate on immutable data. At the least, this ensures that inference results are consistent for a given input.

Regarding availability, we have a daemon in place to detect system downtime at the application level. Currently, it is the responsibility of an operations engineer to manually restart the system whenever it is down, but this can change in the future.

2.6 Privacy

While privacy is a problem for most human genetics related work, fm-tools only works with summary level data, which cannot easily be traced back to the individual. It has been shown that algorithms can work with summary level data to output meaningful results [10].

Chapter 3

Overview of system architecture

Fm-tools consists of several components (Figure 3-1): fm-api-server, fm-database, and fmtk.

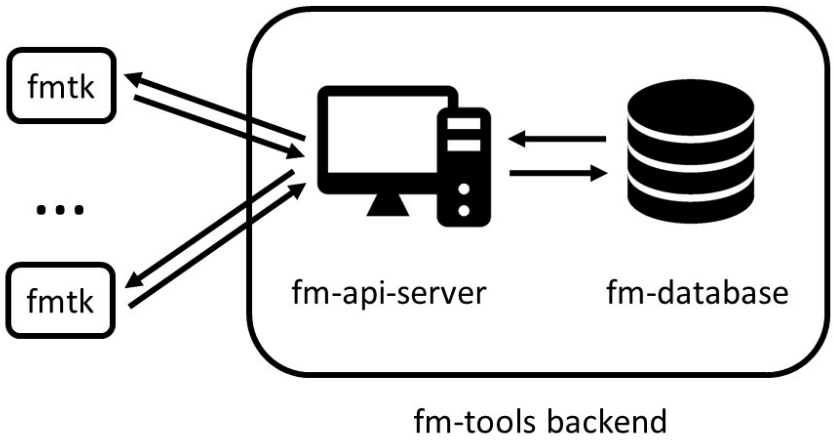


Figure 3-1: Fm-tools consists of fm-api-server, fm-database, and fmtk (the client). Any number of clients can concurrently communicate with the fm-api-server.

3.1 Fm-api-server

The fm-tools API server (fm-api-server) is deployed as an Nginx-uWSGI-Flask application, as shown in Figure 3-2. In this architecture, the API server is implemented based on the Flask framework, uWSGI hosts the API server, and Nginx is the reverse proxy. We choose this architecture because it is naturally scalable with the number of clients: as the number of clients increases, we can introduce more fm-api-server processes under the load balancer to service the clients.

The Nginx and uWSGI configurations are shown in Figure 3-3. The `uwsgi_read_timeout` directive Figure 3-3a is set to 300 seconds to allow lagging operations at the 95th percentile to complete successfully on fm-api-server. Although most API clients time out after 60 seconds by default, we believe that this setting will obviate some unnecessary operations by allowing certain clients to wait for anomalous operations instead of spawning new ones.

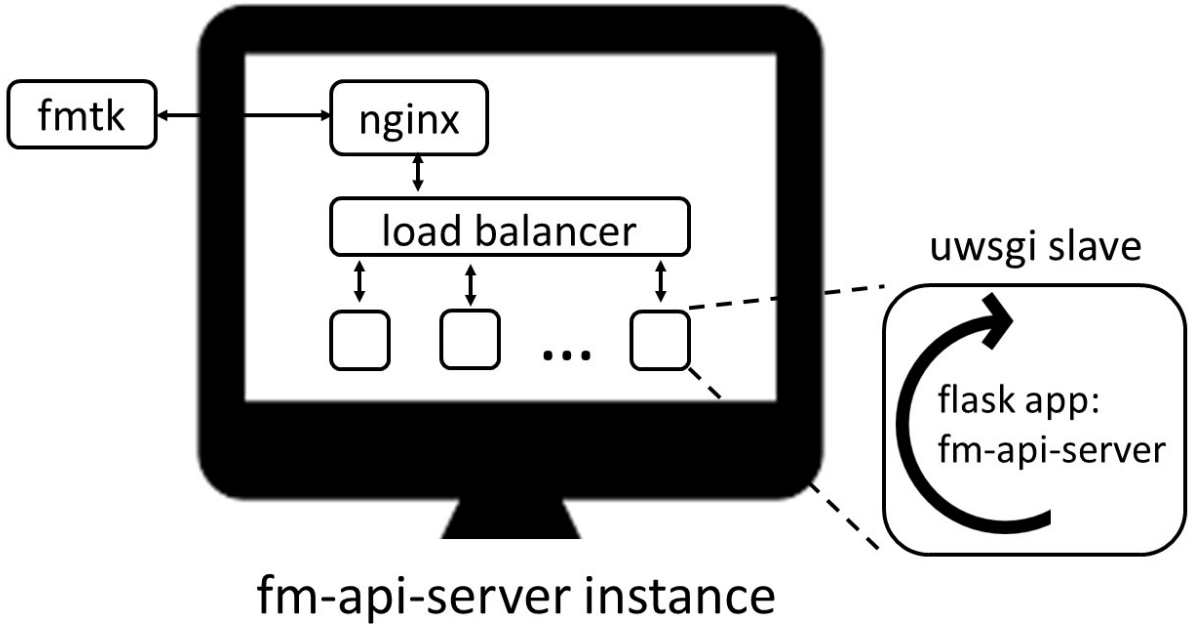


Figure 3-2: Fm-api-server is deployed as an Nginx-uWSGI-Flask application. Note that the number of fm-api-server processes can scale with the number of clients (fmtk instances), and that all of the communication is managed by a load balancer.

Currently, fm-api-server is hosted on an OverStack instance in MIT CSAIL. The instance is

```
server {
    listen 80;
    server_name $hostname;

    location / {
        include uwsgi_params;
        uwsgi_pass unix:/home/ubuntu/haploviz-nucleus/apiserver/apiserver.sock;
        uwsgi_read_timeout 300;
    }
}
```

(a) The Nginx configuration.

```
description "uWSGI server instance configured to serve apiserver"

start on runlevel [2345]
stop on runlevel [!2345]

setuid ubuntu
setgid www-data

env PATH=/home/ubuntu/haploviz-nucleus/apiserver/apiserverenv/bin
chdir /home/ubuntu/haploviz-nucleus/apiserver
exec uwsgi --ini apiserver.ini
```

(b) The uWSGI configuration.

Figure 3-3: The `uwsgi_read_timeout` directive (a) is set to 300 seconds (default 60) to ensure that lagging operations (95th percentile) can complete successfully.

an `xl.2core` instance with 8 GiB RAM, 2 vCPUs, and 64 GIB SSD. The operating system is vanilla Ubuntu 14.04.

Fm-api-server exposes the fm-tools API (described in Section 6) as JSON-RPC, which is a lightweight, flexible, and readable framework. In addition, this framework is widely accepted across web services and allows for easy integration with dedicated visualization applications (discussed in Section 9).

3.2 Fm-database

The fm-tools database (fm-database) is implemented using MySQL 14.14. Currently, fm-database runs as a single-sharded service on the same instance as fm-api-server. To scale with the data volume, we can introduce sharding in the future (the discussion of which is outside the scope of this paper).

3.3 Fmtk and fmpy

End users of fm-tools are provided with the Fine Mapping Toolkit (fmtk), which consists of a set of command-line utilities and an SDK (fmpy) for interacting with the fm-tools backend (fm-api-server and fm-database; Figure 3-1). Fmtk is a lightweight tool that can be installed on any Linux server with Python 2.7 or higher. This minimal requirement can be easily satisfied on a number of computing environments, which makes it easy for general scientists to use fm-tools.

Chapter 4

Data model

The fm-tools data model is implemented by the SQL schema shown in Figure 4-1 on MySQL. Below, we describe several first-class entities in fm-tools.

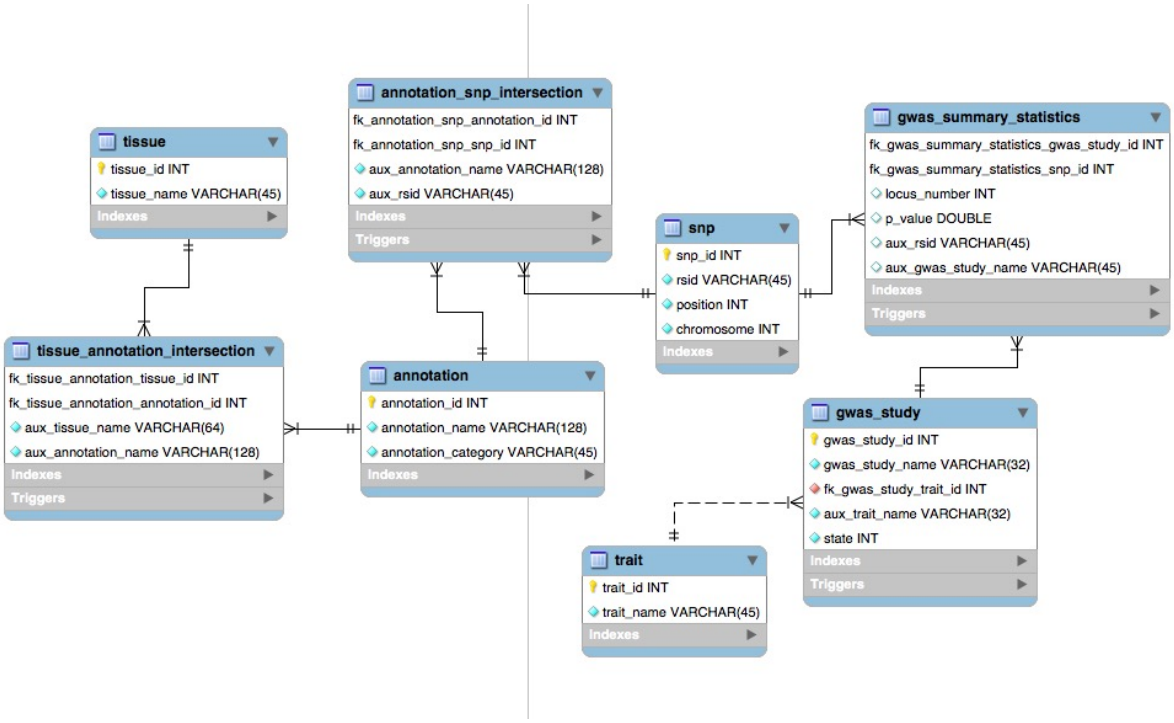


Figure 4-1: The fm-database schema.

4.1 SNPs

In fine mapping, the goal is to identify the causal variants for a given phenotype. In this paper, we focus on single-nucleotide polymorphisms (SNPs) because the current inference engine only considers them (as described in Section 7).

A SNP is a single-base-pair difference across individuals. SNPs can be uniquely identified by their rsids (e.g., rs66702144). In addition, a given SNP can be mapped to a certain chromosome at a certain position along the chromosome, for a given reference genome. Within a reference genome, a given SNP is mapped to a unique position; however, a given SNP may map to two different positions between two different reference genomes.

In light of the above, the `snp` table is designed as shown in Figure 4-1. The primary key (PK) in the table is the rsid of the SNP. In addition, we choose to include a second (system generated) PK to support including the same rsid across different reference genomes.

4.2 Epigenomic annotations

In fm-tools, we aim to support doing iterative fine mapping across a variety of epigenomic contexts. A given epigenomic context is represented by a number of epigenomic annotations. Each epigenomic annotation marks a number of genomic ranges in the reference genome, some of which are active in all cells, while others are active only in cells of certain tissue types. Epigenomic annotations that mark genomic regions active in all cells are denoted by baseline annotations. In contrast, epigenomic annotations that mark regions active only in a certain tissue type are denoted by tissue-specific annotations.

We represent epigenomic annotations as first-class entities in fm-tools. Because there is no universal way to denote a given epigenomic annotation, we choose to use the system generated PK of the annotation table in Figure 4-1 to uniquely identify it. To make it easier for end users to discover and parse the identities of the epigenomic annotations, we additionally support a unique and human-readable name for every epigenomic annotation.

Because many epigenomic annotations can map to a single tissue type, we choose to use two separate tables to establish the mapping. The first table is the `tissue` table, which enumerates the possible tissue types of the epigenomic annotations. The second table is the intersection table `tissue_annotation_intersection`, which maps each epigenomic annotation to its corresponding tissue type, if any, using foreign keys (FKs). Using these two tables to establish the mapping instead of creating a column in the annotation table reduces disk consumption and avoids a SQL anti-pattern [11].

Although a given epigenomic annotation marks a given set of genomic ranges, we choose to omit the genomic ranges of epigenomic annotations in the data model. Instead, we capture the genomic ranges of a given epigenomic annotation in its set of intersections (described in Section 4.3).

4.3 SNP-epigenomic-annotation intersections

At a given region in the genome, there can exist a SNP as well as a number of epigenomic annotations (see Figure 4-2 for an example). If an epigenomic annotation marks the genomic position at which the SNP resides, then we say that the two intersect. For example, suppose a SNP exists in the promoter region of a gene that is only active in the immune tissue type. In this case, we can assume that the SNP intersects with the promoter annotation (which is a baseline annotation) as well as with a number of immune-specific (tissue-specific) annotations.

In fm-tools, SNP-epigenomic-annotation intersections are represented as first-class entities. This is important because, as mentioned above in Section 4.2, we do not explicitly represent the genomic ranges of epigenomic annotations. As a result, fm-tools must use these intersections to resolve the genomic regions that are marked by a given epigenomic annotation. In addition, explicitly modeling the intersections is useful because they are key for implementing the inference engine (see Section 7). Furthermore, explicitly modeling the intersections simplifies the implementation of fm-tools.

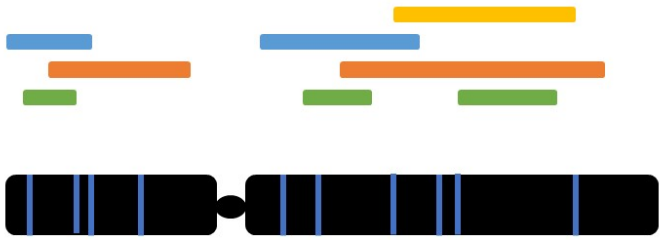


Figure 4-2: A chromosome (black) with SNPs (vertical blue bars) within their respective epigenomic contexts (horizontal bars). As shown, any given SNP can be marked by a number of epigenomic annotations, some of which are baseline annotations and others of which are tissue-specific annotations.

4.4 GWAS summary statistics

The starting point for fine mapping in this paper is the GWAS. Each GWAS pertains to a single disease phenotype, and consists of a number of summary statistics for all genotyped SNPs in the GWAS [10]. Among other things, each summary statistic provides the p-value of association of a genotyped SNP for the corresponding disease phenotype as well as the SNP's locus. A locus is a group of GWAS SNPs that reside in a genomic region that exhibits linkage disequilibrium.

Even though each GWAS maps to a single phenotype, a single phenotype can be mapped to by a number of GWAS. Correspondingly, the same SNP can be described by a number of GWAS for the same phenotype. In addition, the same SNP can be described by two different GWAS for two different phenotypes.

In fm-tools, a number of tables are used to represent a GWAS and its data sets. The

`gwas_study` table in Figure 4-1 represents a single GWAS, keyed by a system generated PK. Because a number of GWAS can map to a single phenotype, we use a FK in `gwas_study` to map a GWAS to its corresponding phenotype in a separate table `trait` (not described).

To represent the summary statistics of a given GWAS, we use the `gwas_summary_statistics` table in Figure 4-1. As mentioned above, each summary statistic in fm-tools corresponds to a single GWAS and a single SNP, so we design `gwas_summary_statistics` with a composite PK that consists of a FK to `gwas_study` and a FK to `snp`. In addition, `gwas_summary_statistics` contains the columns `p_value` and `locus` to denote the p-value of association and locus, respectively, of a GWAS SNP.

Notice in Figure 4-1 that `gwas_study` has a `state` column. This column is intended to implement immutable versus mutable GWAS data sets at the application level. If a GWAS is in the `open` state, then its data sets may be changed at any time. For example, an end user may add summary statistics to the GWAS or update existing summary statistics. This mechanism allows scientists to easily create new GWAS data sets and instantiate them in fm-tools. Once a GWAS is in the `closed` state, however, its data sets are immutable. Any changes to the data sets of a `closed` GWAS must necessarily instantiate a new GWAS. This provides consistency and reproducibility, because the inference engine only operates on `closed` data sets.

4.5 Schema implementation

We used MySQL Workbench to design the database schema shown in Figure 4-1. We started by creating a model for the schema. Within the schema model, we created the tables described above, specifying PKs, FKs, and other columns. We note that the EER diagram utility of MySQL Workbench [12] was especially useful in designing the schema. Not only did the EER diagram provide a good visualization of the current state of the schema, but it also allowed us to identify bugs in the design.

Once the minimum viable product schema was completed, we used the MySQL Work-

bench Forward Engineer utility [13] to instantiate the schema against our fm-database instance.

Chapter 5

Data import

Importing data into fm-database can be done in one of two ways: batch import of curated data sets and data object creation via the API. Here, we discuss the batch import pipeline as well as the implementation experience. In Section 6.1, we will discuss data object creation via the API.

5.1 Pipeline

To enable the fast import of curated data sets into fm-database, we provide a pipeline that parses raw data set files and directly imports the parsed data structures into a specified MySQL database. The pipeline is deployed as a first-class script that can be executed on the command-line of the fm-database instance, provided that the calling user has authorization to the fm-database. When parsing the raw data set files, the pipeline accepts as input a directory that contains the files. The directory structure must abide by the semantics enforced by the pipeline. Once the data structures have been parsed, the pipeline uses the `LOAD DATA LOCAL INFILE` construct [14] to efficiently import the data into the MySQL database specified by the calling user (e.g., fm-database). We choose to use this construct because it is much more efficient than the native SQL `INSERT INTO` construct by several orders of magnitude [15], as shown by empirical evidence in Table 5.1.

Construct	Data set size (MiB)	Run time (s)
`LOAD DATA LOCAL INFILE`	858	208.104
`INSERT INTO` (via MySQL Workbench)	147.948	153.2

Table 5.1: Example run times for two MySQL data import constructs. As shown, the `LOAD DATA LOCAL INFILE` construct is only 35.8% slower than the `INSERT INTO` construct (via MySQL Workbench) against a data set that is 479.9% larger. In fact, we omit showing the performance of the `INSERT INTO` construct against larger data sets because we cannot get any operations to succeed.

5.2 Import of RiVIERA data sets

For fm-database, we used the batch import pipeline discussed in Section 5.1 to import data sets for 27 GWAS and 272 epigenomic annotations across the genome. These data sets were used in the manuscript for RiVIERA [7].

5.3 Pipeline implementation

We encountered a number of problems when implementing the batch import pipeline. Here, we discuss the problems encountered as well as the journey to the current implementation.

5.3.1 MySQL Workbench Data Import Wizard performance

We originally tried to use the MySQL Workbench Data Import Wizard [16] to directly import the curated data sets into fm-database. The problem was that the import operation took a very long time to complete, even for moderate data set sizes (Table 5.1). In addition, many of our import attempts timed out for unknown reasons [17] or crashed due to network connectivity problems. Furthermore, the Wizard provided no indication of progress, which made it hard to detect whether the operation is having problems.

5.3.2 MySQL Server configuration

In light of the problems with the MySQL Workbench Data Import Wizard Section 5.3.1, we opted to use the `LOAD DATA LOCAL INFILE` construct, which is presumed to be much more efficient than the Wizard because it uses SQL statements (Table 5.1; [15]). Unfortunately, the construct did not immediately work, and several changes had to be made to the MySQL Server configuration settings to get the construct to work.

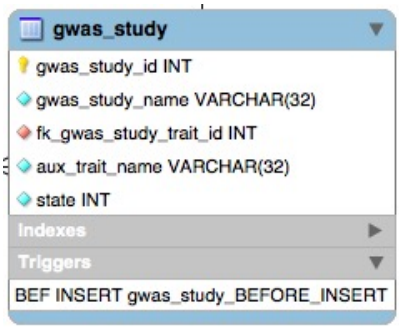
5.3.3 Triggers and FKs

One common problem with importing data into MySQL is establishing FK relationships. One solution is to disable FK constraints in the relevant tables during data import. However, it would have been difficult to programmatically establish the FK constraints post hoc, unless we manually scan through all newly imported rows. Moreover, disabling FK constraints can result in a situation in which data integrity is compromised.

To establish the FK relationships when importing data into fm-database, we use a combination of auxiliary columns and triggers in tables with FKs. Auxiliary columns in a given table are necessary to establish FKs in the given table because we do not know the PKs of other tables at data import time. The auxiliary column should be a descriptor of what the FK should be, which can later be used by a trigger, particularly a `BEFORE INSERT` trigger, at data import time to establish the FK. Just before a row gets inserted into a table with a `BEFORE INSERT` trigger, the trigger can read the auxiliary columns and resolve the appropriate FKs. Once the row *with the FKs* is resolved, the row can be inserted into the table. Most notably, this solution allows the FK constraints to remain enabled throughout the data import operation, thereby ensuring data integrity.

Let us review the `gwas_study` table as an example. Recall that each GWAS maps to exactly one phenotype. This corresponds to the `fk_gwas_study_trait_id` FK of the `gwas_study` table. However, at data import time, we do not know the PK of the phenotype to which the GWAS maps. What we do know is another descriptor of the phenotype: its name. When inserting a row into the `gwas_study` table, we can specify the name of the corresponding

phenotype in the auxiliary column `aux_trait_name` (Figure 5-1a). Just before inserting the row, the `BEFORE INSERT` trigger (Figure 5-1b) resolves the `fk_gwas_study_trait_id` column based on the specified `aux_trait_name`. If a FK is resolved, then the row gets inserted and the data import operation is done. Otherwise, an exception is thrown.



(a)

```
CREATE DEFINER = CURRENT_USER TRIGGER `test2`.`gwas_study_BEFORE_INSERT` BEFORE INSERT ON `gwas_study` FOR EACH ROW
BEGIN
    SET NEW.fk_gwas_study_trait_id = (
        SELECT trait.trait_id FROM trait WHERE trait.trait_name = NEW.aux_trait_name
    );
END
```

(b) The `BEFORE INSERT` trigger.

Figure 5-1: The `gwas_study` table (a) and its `BEFORE INSERT` trigger (b).

Note that this solution is dependent on a few assumptions. First, this solution requires that the phenotype name can be uniquely mapped to a phenotype and its PK. This is enforced by the unique column trait_name in the `trait` table (not shown). Furthermore, this solution requires that the data import occurs in a certain order. As mentioned above, the triggers throw exceptions if no FK can be resolved. This means that the rows that are depended upon by other parent rows must be inserted before the parent rows. Indeed, the ordering is enforced by the pipeline described in Section 5.1, so the running user does not need to be privy to anything.

Chapter 6

Fm-tools API

The fm-tools API (implemented as JSON-RPC) exposes a set of methods for handling and manipulating the first-class data objects described in Section 4. Here, we discuss some of the available APIs and illustrate workflows that can be implemented using the fm-tools API.

6.1 CRU[D] of first-class data objects

The fm-tools API exposes a set of methods for creating, reading, and updating (CRU[D]) the first-class data objects described in Section 4. These APIs can be used as an alternative, and even preferred, way to import data into fm-database. Importing data via the API is preferred because the requesting user does not need to have authorization to fm-database. In addition, the API provides a superset of the consistency guarantees provided by the static data import pipeline in Section 5.1 and Section 5.3.

We highlight the set of APIs available for CRU[D]ing GWAS as a special case of the general API for other data objects. In general, we adopt the HTTP best practices [18] when implementing the API. To create a new GWAS, we expose the

```
post_gwas(name, phenotype_id)
```

method, which implements the HTTP POST request to create a new GWAS with the specified `name`. The phenotype mapped to by the GWAS is determined by the specified `phenotype_id`, which should be a system GUID for the phenotype. The `post_gwas()` method returns the system GUID for the GWAS, which can be used in subsequent requests to manipulate or use the GWAS.

The API ensures that all newly created GWAS are in the `open` state, which means that the GWAS is currently mutable. This can be checked by reading the GWAS using the

```
get_gwas(gwas_id)
```

method.

To add summary statistics to an `open` GWAS, we expose the

```
put_gwas(gwas_id, summary_statistics, close)
```

method. This architecture allows end users to add new summary statistics to a given GWAS asynchronously, which can be useful if the data sets are not all available at one time. This is also useful for allowing end users to fix bugs when importing data. For example, an end user can fix the p-value of association of a SNP if a wrong value was accidentally set at first.

Additionally, the

```
put_gwas(..., close)
```

method allows end users to close a GWAS that is `open`. This effectively converts the GWAS from a mutable data object into an immutable one, thereby preserving all of the relationships that have been established. Because the inference engine only operates on `closed` data objects, this ensures that analyses have a consistent view of the data and can be easily reproduced.

A similar set of APIs is available for manipulating SNPs, epigenomic annotations, and more, but we omit discussing them here.

6.2 Doing inference

In addition to supporting CRU[D] operations, the fm-tools API exposes handles for manipulating the inference engine (discussed in Section 7). Similar to the CRU[D] APIs, these APIs are exposed as JSON-RPCs. However, the design decisions behind the development of these APIs focus more on interactivity, resumability, and performance than on consistency, which we will discuss below.

Figure 6-1 illustrates the general workflow for doing fine mapping analyses. As shown, the inference engine is exposed via two general APIs:

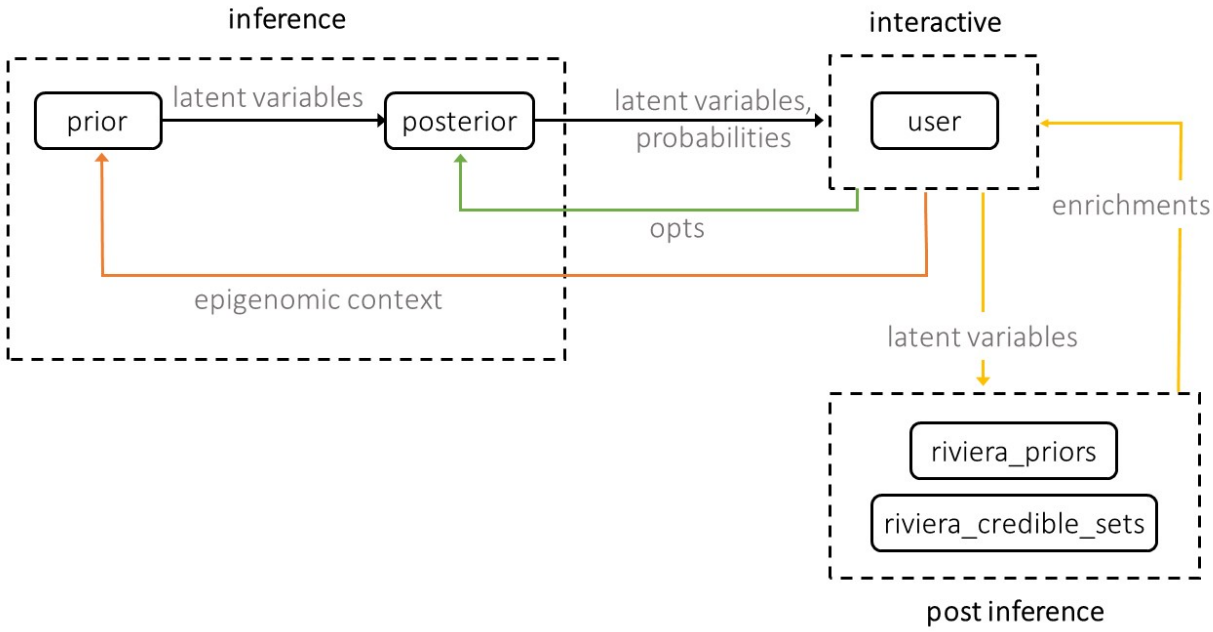


Figure 6-1: The interactive and iterative workflow for doing fine mapping analyses.

`post_fine_mapping_prior(algorithm, gwas_id[, epigenomic_context])`

(henceforth denoted by the prior function) and

`post_fine_mapping_posterior(algorithm, latent_variables[, opts])`

(henceforth denoted by the posterior function). The prior function is responsible for initializing the fine mapping inference operation against a specified data set (identified by the

`gwas_id`) within a specified epigenomic context; that is, it outputs the latent variable estimates that will be used to initialize inference as well as the epigenomic annotations that comprise the context for inference. The requesting user may select from a number of algorithms that will be used to calculate the latent variable estimates. Because this paper focuses on the RiVIERA-mini algorithm discussed in Section 7.2, we will assume hereafter that RiVIERA-mini is the underlying algorithm whenever we refer to the inference operation or the like. In addition, the requesting user may select from a number of epigenomic contexts to use for inference (orange arrow in Figure 6-1). In general, the requesting user may omit specifying a context, thereby implying that all epigenomic annotations will comprise the context for inference. However, this will be an important feature for enabling the requesting user to condition analyses on dynamically generated hypotheses, and we will discuss this in more detail in Section 6.3.

The posterior function is the main function for doing inference. Given the starting latent variable estimates and the epigenomic context from the prior function, the posterior function is responsible for inferring the latent variables that best model the GWAS summary statistics within the specified epigenomic context, based on the algorithm described in Section 7.2. The latent variables inferred by the posterior function can subsequently be used to identify the causal SNPs, as discussed in Section 7.1. Because fine mapping inference is an operation that usually requires batch processing, the inference engine must essentially execute a heavyweight and long running operation that is susceptible to interruptions due to network connectivity problems as well as poor performance.

To ensure the resumability of the posterior function in light of the fact that it encapsulates the long running inference operation, we must break the inference operation down into modular components and expose those components in the API. We begin by defining two new terms: runs and epochs. A run is an execution of the inference operation against some subset of a given GWAS. An epoch is comprised of one or more runs such that the inference operation has been executed against the complete set of the given GWAS. The run length is a configurable parameter, as will be discussed below. The key to resumability is this: instead of exposing an entire epoch (or a number of epochs) via the posterior function API,

we expose only a run. This ensures that the posterior function is lightweight and efficient, because the run length can be easily tuned to yield acceptable latencies. More importantly, each run is fairly short running, so the inference operation can easily be resumed at the last successful run. Moreover, because the inference engine only operates on `closed` data objects, specific runs can be safely repeated if network connectivity problems arise.

One problem with the architecture described so far is that fine mapping inference generally requires hundreds of epochs to complete. Luckily, this is supported by the API in an interactive manner. To complete the inference operation, the requesting user may concatenate multiple runs together into one epoch, and then concatenate multiple epochs together until a sufficient number of epochs is attained. One requirement, though, is that the API must somehow ensure that a run (except the first run) can resume the inference operation where the previous run terminated. This is where the `opts` input of the posterior function comes in. In the output of the posterior function, it indicates whether or not it has completed inference for the current epoch. If the inference for the current epoch is not completed, then the requesting user has the option to resume inference using the `opts`, as depicted by the green arrow in Figure 6-1. With this mechanism, the API clearly provides both resumability and interactivity, but it is still not clear that it ensures performance.

Ensuring performance is a tangential problem. In fact, the architecture described above has the potential of negatively impacting performance because resuming the inference operation does incur overhead cost. To minimize the overhead cost, we can empirically determine the run length that yields the optimal latency for a given GWAS. In the production environment, we use the run length that yields optimal latencies on average across all GWAS (results not shown).

6.3 Dynamically configuring epigenomic contexts

Recall that one of the key features of fm-tools is the ability to configure the epigenomic contexts of fine mapping to reflect dynamically generated hypotheses. To that end, the fm-tools API provides mechanisms for configuring epigenomic contexts both before and after

inference. As with other fm-tools APIs, this set of APIs is also exposed via JSON-RPC. The design decisions here focus on interactivity, usability, and flexibility.

We first discuss configuring epigenomic contexts after inference, or post-inference (Figure 6-1). Suppose that the end user has completed an inference operation consisting of hundreds of epochs for a given epigenomic context, and subsequently aims to explore a new context. The end user may do so by doing the workflow described herein. The output of the last run that was executed by the inference operation contains the latent variable estimates that best allow the inference engine to model the GWAS summary statistics within the given context. Using the post-inference API

```
post_riviera_credible_sets(gwas_id, latent_variables),
```

the end user can get the posterior probabilities of association of the SNPs in the GWAS for the corresponding phenotype, which can subsequently be used to identify the union credible set (as described in Section 7.1; yellow arrow in Figure 6-1). Conditioning on the identity of the causal SNPs in the union credible set, and using a combination of tools provided by fmtk as well as the post-inference API

```
post_riviera_priors(gwas_id, latent_variables),
```

the end user may get the enrichment of every epigenomic annotation within the given context in terms of its responsibility in identifying the causal SNPs (see Section 7.1 for a more detailed description of the semantics of epigenomic enrichments; yellow arrow in Figure 6-1). Given the epigenomic enrichments, the end user may investigate the effects of fine mapping in the context of only the most enriched epigenomic annotations by using the `post_riviera_credible_sets(..., latent_variables)` API and setting the coefficients of all irrelevant epigenomic annotations to zero in the `latent_variables` input. This solution instantly yields re-calculated posterior probabilities of association of the SNPs within what is effectively a new epigenomic context, which consists of only the most enriched epigenomic annotations from the given context. Obviously, this solution provides interactivity, usability, and performance, but most likely at the cost of accuracy.

Luckily, the fm-tools API naturally provides support for re-doing inference in new epigenomic

contexts. As mentioned in Section 6.2, the end user may specify the epigenomic context as input to the prior function to initialize the inference operation only within that context. This mechanism can easily be extended to allow the end user to interactively condition analyses on newly generated contexts, especially after having already completed one inference operation. In particular, an end user may select to re-do inference with a context that consists of the most enriched epigenomic annotations from the previous context. This corresponds to the orange arrow in Figure 6-1.

We reiterate here that fm-tools provides support for dynamically configuring epigenomic contexts both in the inference engine and in the post-inference engine. This provides flexibility for the end user, and allows the end user to select between accuracy and performance when exploring new hypotheses. Nonetheless, the key here is that fm-tools brings the end user's selection to the forefront, thereby providing both flexibility and interactivity.

Chapter 7

The inference engine

This section is dedicated to the fm-tools inference engine, which is currently a component of fm-api-server. Although the inference engine supports a number of fine mapping algorithms and exposes them via the fm-tools API (described in Section 6), this paper focuses on the RiVIERA algorithm. We begin by describing the RiVIERA algorithm for batch processing [7]. Afterwards, we discuss the algorithm implemented in the fm-tools inference engine as well as how it differs from RiVIERA. Finally, we discuss the implementation experience.

7.1 RiVIERA

The RiVIERA algorithm is a Bayesian inference algorithm that simultaneously infers causal SNPs and causal epigenomic annotations for a phenotype, given the GWAS summary statistics for the phenotype and the epigenomic context. Internally, RiVIERA jointly models the specified GWAS p-values of association and the influences of every epigenomic annotation in the specified context, and tries to infer the latent variables that best describe the data. The graphical model of RiVIERA is shown in Figure 7-1.

The prior distribution is a function of the epigenomic context, with the prior probability of association of a SNP v for a phenotype given by

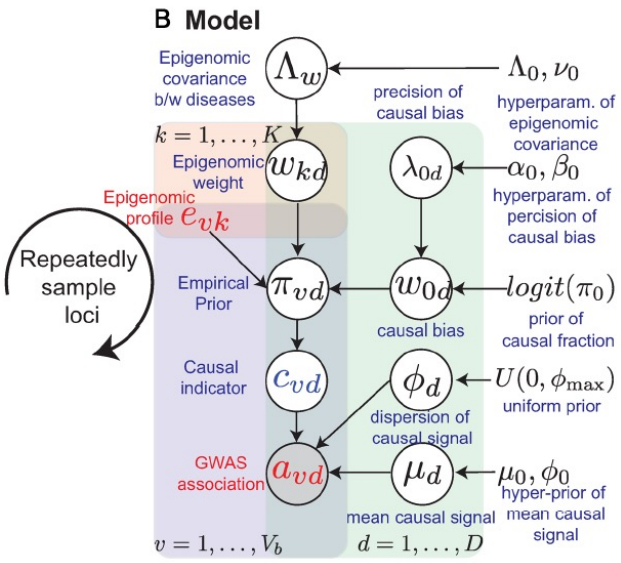


Figure 7-1: The RiVIERA graphical model. This figure is adapted from [7].

$$\pi_v(;\vec{w}, w_0) = \frac{1}{1 + exp(-(\sum_k w_k e_{v,k} + w_0))}, \tag{7.1}$$

where w_k denotes the weight of epigenomic annotation k in describing the phenotype, w_0 denotes the bias epigenomic annotation weight, and $e_{v,k}$ denotes the intersection between SNP v and epigenomic annotation k. Here and elsewhere, we omit the subscripts that denote the phenotype in the corresponding definitions from [7], because this paper focuses on modeling one phenotype at a time. The epigenomic annotation weights $\vec{w} = (w_k)_{k=1}^K$ are modeled as random variables such that

$$w_k \sim \mathcal{N}(0, \Lambda^{-1}) \tag{7.2}$$

and

$$\Lambda \sim \mathcal{W}(1, 0) \tag{7.3}$$

The bias epigenomic annotation weight is modeled as a random variable such that

$$w_0 \sim \mathcal{N}(logit(\pi_0), \lambda^{-1}) \tag{7.4}$$

and

$$\lambda \sim \Gamma(\alpha_0, \beta_0), \tag{7.5}$$

where π_0 (a hyperparameter) denotes the expected percentage of causal SNPs across all loci *a priori*, and α_0 and β_0 are also hyperparameters. Lastly, the intersections matrix $E = (e_{v,k})_{v=1,k=1}^{V,K}$ is modeled as an observed variable: $e_{v,k} = 1$ if SNP v intersects with epigenomic annotation k, and $e_{v,k} = 0$ otherwise.

The likelihood (beta density) function encapsulates the GWAS summary statistics, with the likelihood of a GWAS p-value a_v given by

$$P(a_v; \mu, \phi) = \frac{\Gamma(\phi)}{\Gamma(\mu\phi)\Gamma((1-\mu)\phi)} a_v^{\mu\phi-1}(1-a_v)^{(1-\mu)\phi-1}, \tag{7.6}$$

where μ and ϕ are latent variables in the model. The distributions for μ and ϕ are

$$\mu \sim \mathcal{B}(\mu_0, \phi_0) \tag{7.7}$$

and

$$\phi \sim \mathcal{U}(0, \phi_{max}), \tag{7.8}$$

respectively, where μ_0, ϕ_0, and ϕ_{max} are hyperparameters.

The posterior probability of association (PPA) of a SNP v (in GWAS locus l) for a phenotype is given by

$$P(c_v; \Theta, \theta) = \frac{\pi_v(; \vec{w}, w_0) P(a_v; \mu, \phi)}{\sum_{v' \in V_l} \pi_{v'}(; \vec{w}, w_0) P(a_{v'}; \mu, \phi)}, \tag{7.9}$$

where $\Theta = (\vec{w}, w_0, \mu, \phi)$ denotes the set of latent variables (modulo the hyperparameters), θ denotes the set of hyperparameters, π_v is defined in Equation 7.1, $P(a_v; \mu, \phi)$ is defined in Equation 7.6, and V_l denotes the set of SNPs on locus l. The credible set $\mathcal{C}_l$ for a locus l is defined as the smallest possible set of SNPs in locus l such that the sum of the PPAs of the SNPs contained in the set is greater than or equal to a specified confidence. For example, the sum of the PPAs for the 95% credible set must be greater than or equal to 0.95.

The main assumption in RiVIERA is that every GWAS locus has one causal SNP. This is encapsulated in the model via the representation of the PPA shown in Equation 7.9, which is normalized across all SNPs within a locus.

Because doing inference with the joint posterior distribution of the latent variables given the data (not shown) is infeasible, RiVIERA uses MCMC methods, particularly Gibbs sampling [19] and HMC [20], to sample from the distribution and do inference. Gibbs sampling is used to sample estimates for Λ and λ, and HMC is used to sample estimates for Θ.

Because RiVIERA is a batch processing algorithm, it does inference for a number of epochs (distinct from the epochs described in the fm-tools API in Section 6) before returning inference results to the end user. For each epoch, the RiVIERA algorithm randomly selects a locus and does inference against the data sets that pertain to the locus. One constraint in the RiVIERA model is that the epigenomic weights must be nonnegative across all epochs.

Lastly, we discuss the epigenomic enrichments as defined by RiVIERA. After an inference operation is completed, the RiVIERA algorithm returns the inferred latent variables $\hat{\Theta}$ in the output. In particular, the end user has access to the epigenomic weights ($\hat{\vec{w}}$). However, it is ill-advised to interpret the epigenomic weights directly due to the correlations between the epigenomic annotations. To work around this problem, RiVIERA defines epigenomic enrichments as

$$f_k = log\big(\frac{1}{|\mathcal{C}|}\sum_{v\in\mathcal{C}}\frac{\pi_v(;\hat{\vec{w}},\hat{w_0})}{\pi_v(;w_k=0,\hat{\vec{w}}_{\setminus\hat{w_k}},\hat{w_0})}\big), \tag{7.10}$$

where $\mathcal{C} = \cup_{l=1}^{L}\mathcal{C}_l$ is the union credible set for the phenotype whose GWAS contains L loci. Essentially, the algorithm defines the enrichment of an epigenomic annotation as the ratio between the inferred model and the null model for that annotation, across the causal SNPs.

7.2 RiVIERA-mini

We implement a modified RiVIERA algorithm (RiVIERA-mini) in fm-tools for a number of reasons. The primary reason is that RiVIERA does not support the interactive exploration of dynamically generated hypotheses, which is one of the key deliverables of fm-tools. In addition, RiVIERA is designed for batch processing, which is not amenable to the performance, resumability, and interactivity properties of fm-tools.

In this section, we highlight the ways in which RiVIERA-mini differs from RiVIERA. Note that a majority of the design decisions are made to further interactivity or performance.

First, RiVIERA-mini does not model the variance parameters of the epigenomic weights (Λ, λ) as random variables. In effect, this converts Equation 7.2 into

$$w_k \sim \mathcal{N}(0,1) \tag{7.11}$$

and Equation 7.4 into

$$w_0 \sim \mathcal{N}(logit(\pi_0), 16). \tag{7.12}$$

We do this primarily to simplify the implementation. This also marginally improves performance of inference, because we can omit inferring a few random variables. Lastly, the

importance of the variance parameters is diminished because RiVIERA-mini only models one phenotype at a time (discussed next). We show in Section 7.3.4 that this design decision does not compromise the ability of the inference engine to reproduce the results of RiVIERA.

Second, we only do inference with one phenotype at a time. Recall that the RiVIERA algorithm leverages correlations between related phenotypes to increase predictive power. While simultaneously inferring multiple phenotypes is useful in batch processing, we are more concerned with performance and interactivity in fm-tools. Modeling on one phenotype at a time decreases the number of parameters that must be inferred, thereby allowing us to optimize for performance.

Third, instead of randomly sampling a locus before every HMC sampling, which is what RiVIERA does, RiVIERA-mini deterministically iterates through all loci in order. We do this to ensure that each epoch consists of a consistent number of runs, for a given GWAS. This is important for decreasing the variance of run times across different inference operations.

Fourth, RiVIERA-mini implements HMC sampling using a Stan model with the NUTS sampler [21] instead of directly implementing HMC sampling using gradients. The primary reason for this is to optimize for performance. The NUTS sampler is an optimization and a special case of HMC sampling that has been shown to be orders of magnitude more efficient than HMC sampling in practice. In addition, inference in Stan has been highly optimized over the years, making it the ideal candidate for implementing the inference algorithm in fm-tools.

7.3 Inference engine implementation

We encountered a number of problems when implementing the inference engine. In this section, we discuss the problems encountered, the solutions, and the journey to the current implementation.

7.3.1 One epoch is too much

We originally tried to expose an epoch via the API, but that immediately elicited interactivity problems. The run times of the API requests that pertain to doing inference were on the order of minutes. This amount of latency is unacceptable in a client-server architecture, as most clients time out after about one minute. In addition, most deployment frameworks restrict server response times to one minute. If the server does not respond within the allotted time, then the framework times out with an HTTP 504 status code. Furthermore, the high latency is bad for interactivity, which is one of the key properties of fm-tools. As a result, we opted to expose runs via the API instead of epochs.

7.3.2 Inference implementation with runs

While implementing the inference engine with runs is a neat idea, it does come with some challenges. The main problem pertains to pagination: we want to be able to terminate inference at the end of one run and resume inference in the subsequent run exactly where we terminated. This means that we must ensure that the inference output that results from passing through a state is the same as the output that results from starting at that state (maybe because of termination for a run). For this, we rely on the ergodicity property of Markov chains. We omit the proof of ergodicity in this paper.

To be able to leverage the ergodicity property when implementing the inference engine, we must somehow save the current state of the inference operation when we terminate for a run. Our initial attempt to do this was flawed. Recall from Section 7.1 that RiVIERA (and hence RiVIERA-mini) does HMC sampling against the data sets of one given locus at a time. The output of the HMC sampling consists of latent variable estimates that best model the data sets of that given locus, which we denote by the local estimates. What we initially did to implement the inference engine was output the local estimates as the current state of the inference operation. We then specified these local estimates along with other parameters as the `opts` input to the posterior function to resume the inference operation (green arrow in Figure 6-1). Unfortunately, this yielded highly unstable and inaccurate

results (not shown).

The solution to this problem was to note that both the RiVIERA and the RiVIERA-mini algorithms attempt to infer *global* latent variable estimates, not local. To rectify the bug described above, we established the global estimates as well as the sufficient statistics for calculating the estimates. These global estimates and the sufficient statistics can be outputted at the end of one run and specified as the `opts` input to the posterior function (green arrow in Figure 6-1) to correctly resume the inference operation. With this, we were able to obtain the results shown in Section 7.3.4.

7.3.3 Stan bug

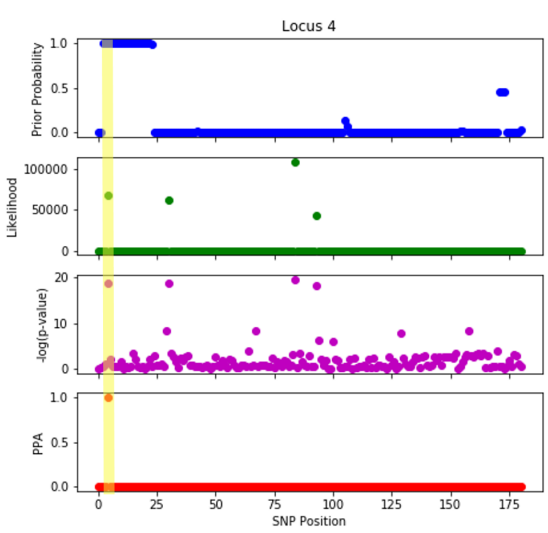
One obstacle that we encountered while implementing the inference engine was a Stan bug regarding numeric overflow during sampling. The bug manifested when we used the *tgamma* function with a bad input that was outside of the domain of the function. Because `tgamma` was being used in sampling, Stan ideally should have rejected the sample, instead of crashing. After working with the current Stan developers to triage the bug, we were notified that the patch would not be deployed for quite some time [22]. In the end, we worked around this problem by directly implementing the buggy parts of Stan.

7.3.4 Reproduction of RiVIERA results

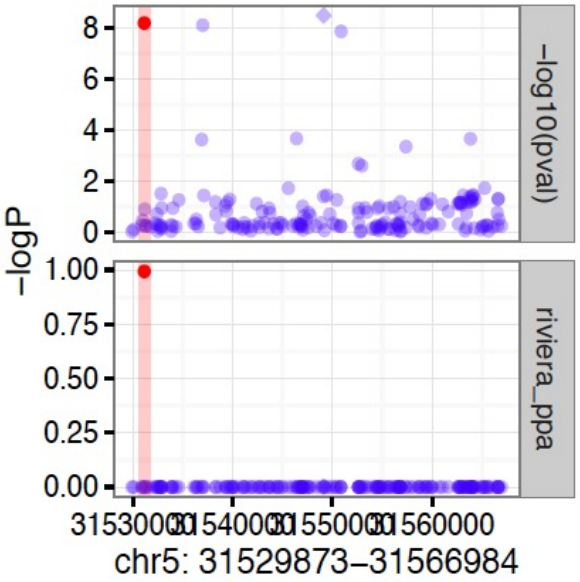
To assess our implementation of RiVIERA-mini, we compared its performance with RiVIERA against the simulation data set that was used in the RiVIERA manuscript [7]. Hereafter, we refer to the simulation data set as the known GWAS. In addition to containing the type of data sets that other GWAS contain, the known GWAS contains the identities of the causal SNPs as well as the true epigenomic annotation weights.

The Figure 7-2 shows the inference results for one locus in the known GWAS (the results for the other loci can be found in Section 11.1). As shown, the results of RiVIERA-mini and RiVIERA are similar: the causal SNP could be identified in a locus that exhibits mild

linkage disequilibrium (i.e., a number of SNPs on the locus were assigned significant p-values by the GWAS; purple in Figure 7-2a; top in Figure 7-2b) by leveraging information from the epigenomic context (blue in Figure 7-2a). Although we only compare the results for this one locus, we affirm that the results for other loci are similar between RiVIERA-mini and RiVIERA. More importantly, the same set of causal SNPs were identified by both algorithms. In fact, there is only one locus (Locus 3) whose causal SNP could not be identified (Figure 7-3). We postulate that that is because Locus 3 exhibits strong linkage disequilibrium (purple in Figure 7-3). However, note that both algorithms (RiVIERA not shown) did infer a high prior probability of association for the causal SNP that could not be implicated (blue in Figure 7-3), which shows that the epigenomic context indeed increases the predictive power of the algorithms.



(a) Prior probabilities of association of SNPs (blue), likelihoods of p-values (green), negative-logarithms of p-values (purple), and PPAs of SNPs (red) for simulated Locus 4 according to RiVIERA-mini. The known causal SNP is indicated by the yellow bar.



(b) Negative-logarithms of p-values (top) and PPAs of SNPs (bottom) for simulated Locus 4 according to RiVIERA. The known causal SNP is indicated by the red bar. This figure is adapted from [7].

Figure 7-2: Inference results for one locus (Locus 4) in the known GWAS.

The Figure 7-4 shows the epigenomic annotation enrichments inferred by both algorithms relative to the true epigenomic annotation weights. Again, we highlight that both RiVIERA-

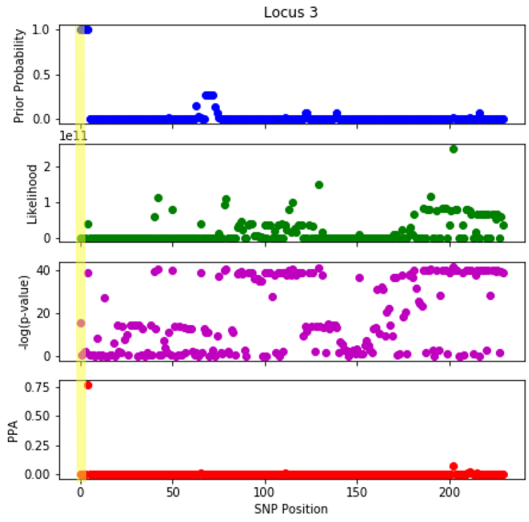


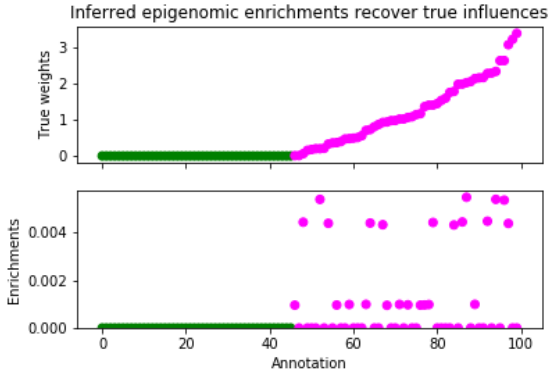
Figure 7-3: Inference results for one locus (simulated Locus 3) in the known GWAS. Although the inference engine is unable to identify the causal SNP (the PPA (red) is low), it is able to leverage the epigenomic context to infer a high prior probability of association (blue) to the causal SNP.

mini and RiVIERA yield similar results. As described in Section 6.3, an end user may use these enrichments to condition subsequent analyses by specifying new epigenomic contexts.

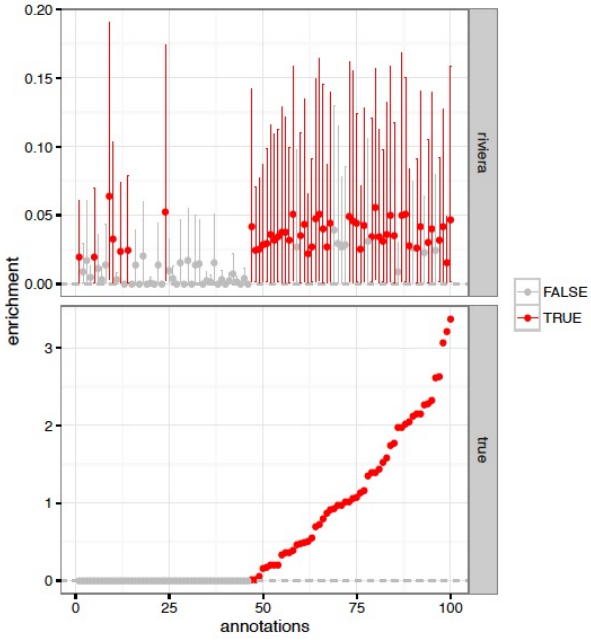
Table 7.1 compares the other latent variables that were inferred by the two algorithms. As shown, ϕ and w_0 were nearly identical between the two algorithms. However, the inferred μ values differed. We hypothesize that further tuning of the hyperparameters may be needed, but we neglected to do so because we were able to reproduce the primary results of RiVIERA despite the discrepancy with μ.

	μ	ϕ	w_0
RiVIERA-mini	0.301	1.569	-8.656
RiVIERA	0.070	1.367	-9.011

Table 7.1: A comparison of the inferred latent variables between RiVIERA-mini and RiVIERA. Although the inferred μ values differed, the inferred ϕ and w_0 values were nearly identical. Further hyperparameter tuning in RiVIERA-mini may be needed for μ.



(a) True epigenomic annotation influences (top) and inferred epigenomic annotation enrichments (bottom) for the 70% credible set.



(b) Inferred epigenomic annotation enrichments (top) for the 95% credible set and true epigenomic annotation influences (bottom).

Figure 7-4: RiVIERA-mini (a) and RiVIERA (b) accurately recover the true epigenomic annotation influences in the known GWAS.

Chapter 8

Case study of celiac disease

It is well known that celiac disease is a hereditary autoimmune disorder that affects the small intestine. Individuals with celiac disease often exhibit gastrointestinal problems such as diarrhea and malabsorption. To understand the genetic causes for celiac disease, a scientist by the name of Alice aims to use fm-tools to interactively explore different hypotheses in order to identify SNPs and epigenomic contexts that are likely to be influential in disease pathogenesis.

Alice begins her analysis by selecting the celiac disease GWAS exposed by fm-tools. She verifies that the GWAS contains several SNPs of interest by using the fm-tools API.

After she is satisfied with the celiac disease GWAS, Alice initiates a fine mapping operation against fm-api-server. Because she does not yet have any hypothesis regarding the epigenomic context, she decides to do the fine mapping operation within the complete epigenomic context (i.e., all epigenomic annotations in fm-database will comprise the context).

After completing 128 epochs of inference, Alice obtains the results shown in Figure 8-1a for one locus (the results for other loci are not shown). Based on the inferred latent variables, she additionally obtains enrichments for every epigenomic annotation that comprised the complete context (Table 8.1). As shown, the immune tissue type was highly enriched in the union credible set identified by inference. Interestingly, the gastrointestinal tissue type was also fairly enriched, which is consistent with the fact that individuals with celiac disease

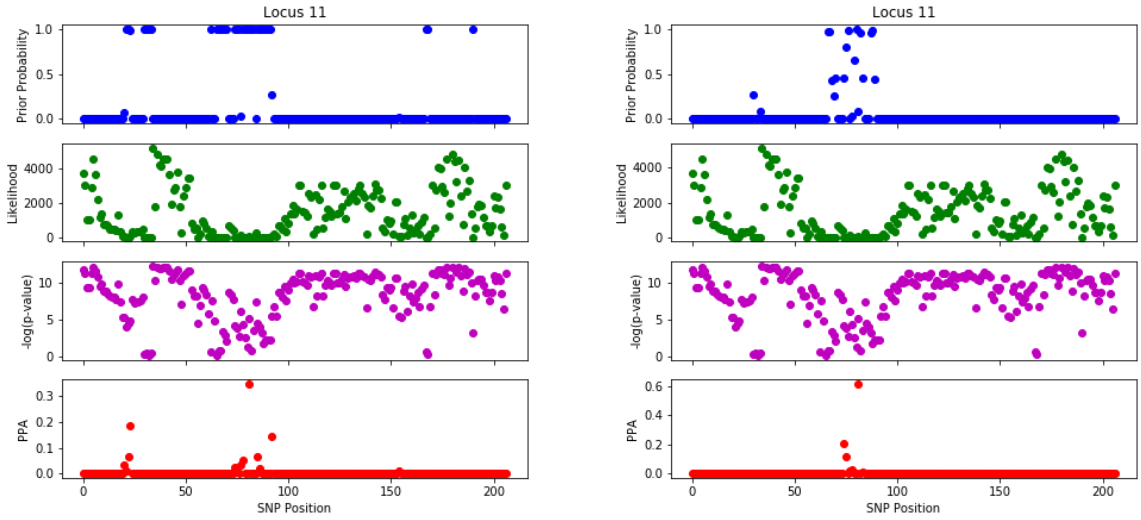
often exhibit intestinal symptoms.

Tissue Type	**Maximum Enrichment**
Immune	0.4944
Central Nervous System	0.4809
Gastrointestinal	0.4634
Skeletal Muscle	0.4003
Adrenal/Pancreas	0.3983
Liver	0.3519

Table 8.1: The maximum enrichment values of epigenomic annotations grouped by tissue type for the celiac GWAS, according to RiVIERA-mini. Immune-specific annotations exhibit the highest enrichment values, as expected for an autoimmune disease.

Happy with the results, Alice decides to re-run the inference operation, but this time she has a hypothesis: Alice believes that only the most highly enriched epigenomic annotations are actually influential in celiac disease. She re-runs the inference operation within a context that consists of just the epigenomic annotations in the 25th percentile of the enrichments from the first operation. The results of the second inference operation are shown in Figure 8-1b. The results here do not differ much from the first operation, but the inference engine does seem to have more confidence in its predictions.

With the results of her analyses in hand, Alice goes for a short break before discussing the results with her PI.



(a) Complete epigenomic context.

(b) Subset epigenomic context.

Figure 8-1: Prior probabilities of association of SNPs (blue), likelihoods of p-values (green), negative-logarithms of p-values (purple), and PPAs of SNPs (red) for one locus (Locus 11) of the celiac GWAS according to RiVIERA-mini. Doing inference against the celiac GWAS within the complete epigenomic context (a) identifies a large number of SNPs with moderate PPAs. Re-doing inference within a context that consists of only the most enriched annotations from the first operation (b) yields fewer SNPs with higher PPAs.

Chapter 9

Limitations and future work

In this section, we briefly mention the current limitations of fm-tools as well as future work that we envision for it.

9.1 Caching

Fm-tools currently does not provide any caching, which can be a problem for interactivity. There are two ways in which fm-tools can provide caching. The first way is that it can provide caching at the inference level. When an end user is attempting to do inference with a given input, fm-api-server can first consult a cache to check whether that particular inference has been done before. If it has, then fm-api-server can return the cache results immediately, greatly improving interactivity and performance.

The second, and more important, way is that fm-tools can provide caching at the latent variables level. What this means is that fm-database can store latent variable estimates, particularly epigenomic annotation weights, that can be used to immediately calculate inference results for any given epigenomic context. Although fm-tools does provide a mechanism for doing so, as described in Section 6.3, it does require that at least one inference operation has been done.

The key to being able to cache at the latent variables level is conditional independence. That is, each latent variable estimator must be conditionally independent of the other estimators, given the data. One idea for achieving this is to estimate the epigenomic annotation weights using correlations: for a given epigenomic annotation, we can calculate the correlation of its (binarized) activity across the genome with the p-values of GWAS SNPs across the genome. These epigenomic annotation weight estimators are conditionally independent because the correlations naturally marginalize the dependencies across the genome. We omit the discussion regarding estimators for the other latent variables, but we postulate that we can obtain good estimators for them.

Fm-tools can provide caching at the latent variables level by calculating the estimators at GWAS close time, after which the GWAS and its data sets are immutable. Although this increases the run time of closing a GWAS, it can yield far greater gains for the run time of inference, which is good for interactivity.

9.2 Job manager and batch processing

Although the focus of fm-tools is interactivity, we acknowledge that support for batch processing is both necessary and useful. To achieve this in a user-friendly way, we propose implementing a job manager that can queue and monitor non-interactive jobs on fm-tools. With this, scientists can opt to spawn long running jobs with hundreds of epochs of RiVIERA-mini to obtain results with high accuracy after they have sufficiently explored a number of hypotheses interactively.

9.3 Others

We acknowledge that inference stability on fm-tools currently varies by GWAS. This is an artifact of RiVIERA-mini not modeling the variance parameters as random variables (see Section 7.2). To alleviate this problem, more work must be done to fine tune the parameters

of the inference engine to optimize inference in the common case. The support for batch processing mentioned in Section 9.2 will also help. In the meantime, the fm-tools API does allow for long running inference operations with hundreds of epochs, so end users can select between short running and interactive jobs, and long running jobs that yield more accurate results.

Resource contention is a problem in fm-tools, primarily because the inference engine is currently deployed alongside fm-api-server. Each inference operation consumes a large amount of processing power and memory, so having concurrent inference operations on the same server can drastically impact performance. In the future, we aim to investigate ways to isolate the inference engine from the fm-api-server so that resource contention can be alleviated, and so that we can horizontally and dynamically scale the inference engine to satisfy time-varying loads.

Recall that one of the target audiences of fm-tools is machine learning researchers. In particular, fm-tools aims to provide a platform for research and development. Although researchers can implement and test their algorithms directly on the inference engine, it does require that the researchers have authorization to the engine and that they understand the semantics of the engine. One solution to this is to explore ways to expose lambda functions via the API so that researchers can specify custom functions to run against the harmonized data store provided by fm-database.

One other limitation of fm-tools is that it does not include a visualization framework. Although there does exist a primitive web application for visualization [23], much work must be done to ensure that the workflows are intuitive and interactive. In addition, the plethora of data must be visualized in such a way that can easily be parsed by scientists so that they can do analyses, which brings us to our final point.

In the future, we aim to extend the existing fm-tools API to support pathway analysis. The current API is limited to allowing scientists to do analyses at the nucleotide (SNP) and gene regulatory (epigenomic) levels. However, it will be more useful for scientists to be able to go even further: to the gene and pathway levels. We believe that this will enable a more end-to-end workflow for analyzing disease pathogenesis as well as for identifying targets for

therapeutic development.

Chapter 10

Conclusion

In conclusion, fm-tools provides an end-to-end solution for fine mapping. It is a platform that allows scientists to do interactive analyses as well as to explore dynamically generated hypotheses. In addition, it is a research and development platform for machine learning researchers to develop and test new algorithms for fine mapping. Furthermore, fm-tools provides a harmonized data store that is exposed via a rich API that can be plugged into a wide variety of other services (e.g., web applications for visualization). In its current state, fm-tools standardizes the way that fine mapping is done and reduces the overhead for developing new machine learning algorithms for fine mapping. Although it is a modest start, we believe that fm-tools represents the first of many steps towards a future in which personalized medicine is a reality.

Bibliography

[1] Slatkin M. Linkage disequilibrium - understanding the evolutionary past and mapping the medical future. *Nature Review Genetics*. **9**, 477-485 (2008).

[2] Ward LD and Kellis M. Interpreting non-coding genetic variation in complex traits and human disease. *Nature Biotechnology*. **30**, 1095-1106 (2012).

[3] The ENCODE Project Consortium. An integrated encyclopedia of DNA elements in the human genome. *Nature*. **489**, 57-74 (2012).

[4] Ward LD and Kellis M. HaploReg: a resource for exploring chromatin states, conservation, and regulatory motif alterations within sets of genetically linked variants. *Nucleic Acids Research*. **40**, 930-934 (2012).

[5] Pickrell JK. Joint Analysis of Functional Genomic Data and Genome-wide Association Studies of 18 Human Traits. *The American Journal of Human Genetics*. **94**, 559-573 (2014).

[6] Chung D *et al*. GPA: A Statistical Approach to Prioritizing GWAS Results by Integrating Pleiotropy and Annotation. *PLOS Genetics*. 10(**11**) (2014).

[7] Li, Y and Kellis, M. Joint Bayesian inference of risk variants and tissue-specific epigenomic enrichments across multiple complex human diseases. *Nucleic Acids Research*. 44(**18**) (2016).

[8] Gilbert S and Lynch NA. Perspectives of the CAP theorem. *Computer*. **45**, 30-36 (2012).

[9] Herlihy MP *et al*. Linearizability: a correctness condition for concurrent objects. *ACM TOPLAS*. **12**, 463-492 (1990).

[10] Pasaniuc B and Price AL. Dissecting the genetics of complex traits using summary association statistics. *Nature Review Genetics*. **18**, 117-127 (2017).

[11] Karwin B. *SQL antipatterns*. Pragmatic Bookshelf (2010).

[12] Oracle Corporation. Adding an EER diagram. (https://dev.mysql.com/doc/workbench/en/wb-creating-eer-diagram.html). (2017).

[13] Oracle Corporation. Forward engineering to a live server. (https://dev.mysql.com/doc/workbench/en/wb-forward-engineering-live-server.html). (2017).

[14] Oracle Corporation. Load data local infile syntax. (https://dev.mysql.com/doc/refman/5.7/en/load-data.html). (2017).

[15] Why is load data infile faster than normal insert statements. (https://dba.stackexchange.com/questions/16809/why-is-load-data-infile-faster-than-normal-insert-statements). (2012).

GitHub: (`https://github.com/yueli-compbio/RiVIERA-beta`).

[16] Oracle Corporation. SQL data export and import wizard. (https://dev.mysql.com/doc/workbench/en/wb-admin-export-import-management.html). (2017).

[17] MySQL Workbench keeps crashing on Mac Mountain Lion. (https://bugs.mysql.com/bug.php?id=68080). (2013).

[18] Richardson L and Ruby S. *RESTful web services*. O'Reilly Media (2007).

[19] Geman S and Geman D. Stochastic relaxation, Gibbs distributions, and the Bayesian restoration of images. *IEEE TPAMI*. **6**, 721-741 (1984).

[20] Neal RM. MCMC using Hamiltonian dynamics. *Handbook of Markov Chain Monte Carlo*. (2011).

[21] Hoffman MD and Gelman A. The No-U-Turn sampler: adaptively setting path lengths in Hamiltonian Monte Carlo. *arXiv:1111.4246* (2011).

[22] Numeric overflow, tgamma. (https://github.com/stan-dev/stan/issues/2255#issuecomment-288497719). (2017).

[23] Haploviz. (https://github.com/a-newman/haploviz). (2017).

Chapter 11

Appendix

11.1 RiVIERA-mini reproduction of RiVIERA

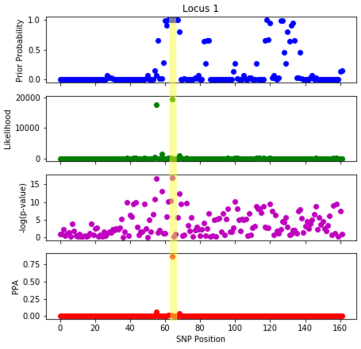


Figure 11-1: Inference results for one locus (simulated Locus 1) in the known GWAS.

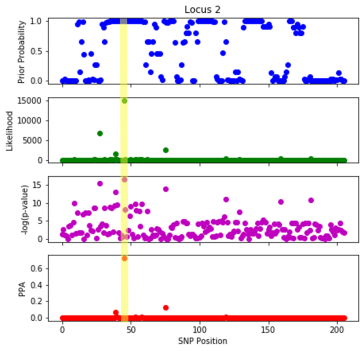


Figure 11-2: Inference results for one locus (simulated Locus 2) in the known GWAS.

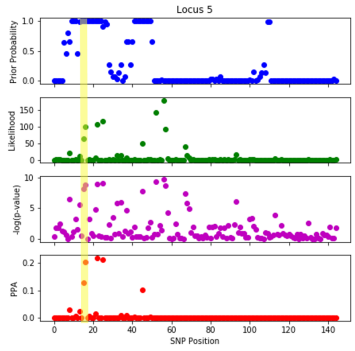


Figure 11-3: Inference results for one locus (simulated Locus 5) in the known GWAS.

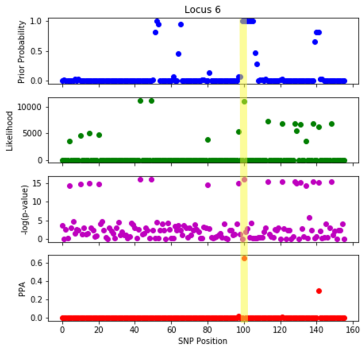


Figure 11-4: Inference results for one locus (simulated Locus 6) in the known GWAS.

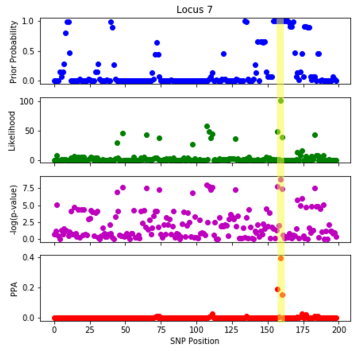


Figure 11-5: Inference results for one locus (simulated Locus 7) in the known GWAS.

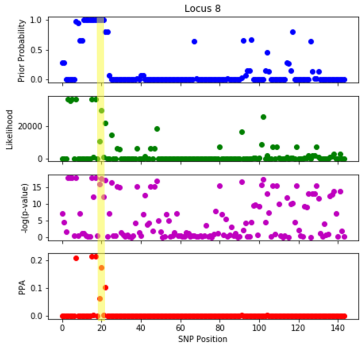


Figure 11-6: Inference results for one locus (simulated Locus 8) in the known GWAS.

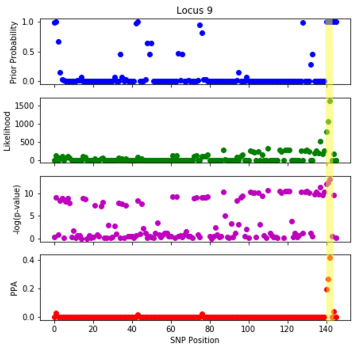


Figure 11-7: Inference results for one locus (simulated Locus 9) in the known GWAS.

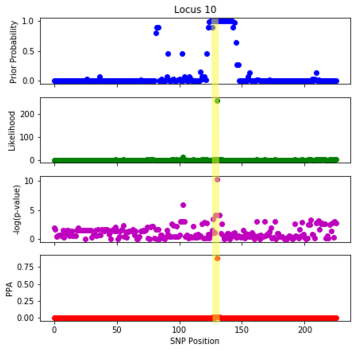


Figure 11-8: Inference results for one locus (simulated Locus 10) in the known GWAS.

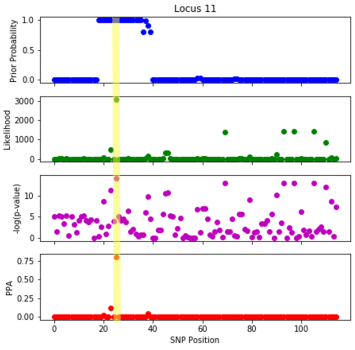


Figure 11-9: Inference results for one locus (simulated Locus 11) in the known GWAS.

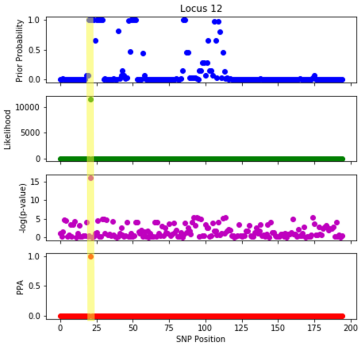


Figure 11-10: Inference results for one locus (simulated Locus 12) in the known GWAS.

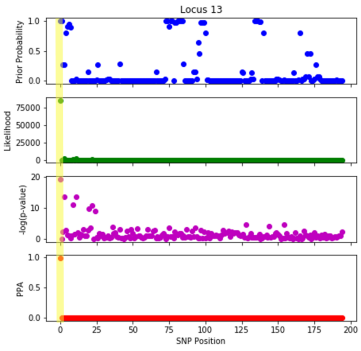


Figure 11-11: Inference results for one locus (simulated Locus 13) in the known GWAS.

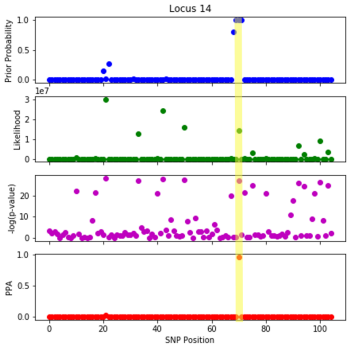


Figure 11-12: Inference results for one locus (simulated Locus 14) in the known GWAS.

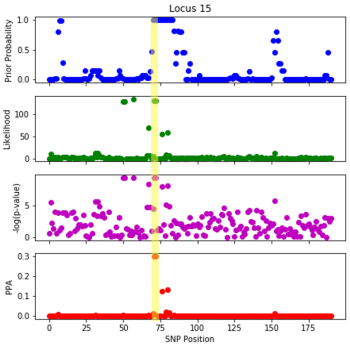


Figure 11-13: Inference results for one locus (simulated Locus 15) in the known GWAS.