AVPPIN Software Architecture

1. Context

The **A**nalysis and **V**isualization of **P**rotein-**P**rotein **I**nteraction **N**etworks (**AVPPIN**) tool allows users to store, compare, and visualize protein-protein interaction (PPI) networks on the web. This tool will enable population genetics of PPI networks between affected and unaffected individuals across multiple data sets to find new insights for disease. Users can access AVPPIN from the web and create their own accounts.

The purpose of this document is to:

- Define the key functions, organization, software components, database, and security model for AVPPIN. **These are draft guidelines.**
- Satisfy requirements for BMSE Assignment 2
- Provide a development guide for a similar system at the Bin Zhang Lab @ ISMMS

2. Functional overview

I. What does the system do?

The AVPPIN will provide many functional features to enable the efficient analysis of PPI networks:

- Ingestion of PPI networks from a variety of formats into a common standard
  - Extracting PPI network files from a variety of formats (e.g. .txt, .gml, .cys)
  - Detect improperly formatted files, including unknown fields and proteins
  - Transforming PPI networks into a standard vocabulary (e.g. Ensembl ID) to facilitate comparison from a large set of alternate vocabularies (e.g. gene name)

- Storage of PPI network data in a relational database (e.g. MySQL) to create a data warehouse for future research
  - Loading PPI networks into the relational database indexed by an ID that facilitates future analysis
  - Includes both public (accessible to all users) and private data (accessible to a specific user(s))

- Annotating PPI networks with Gene Ontology (GO) information to understand PPI network functional significance
  - GO classifies groups of proteins as belonging to certain families by functional pathway or enrichment

- Comparison of PPI networks across datasets using statistical functions to find new insights about health and disease
  - These may be coded in *at least* either Python and R
  - “Does an affected network differ statistically from an unaffected network? Is a sub-network of proteins that differs between an affected network and an unaffected network over or under-represented in protein classification categories? Other analyses will be added as needed.”

- Ingestion and storage of study and network metadata, indexed by one or more ID(s)

In addition, the AVPPIN will provide a web-browser based interface where users may access all of the mentioned functions as well as the following:

- Create user accounts, log in and out, upload and download data, load statistical functions, and perform, save, and view analyses
- Visualize gene networks in the web browser

II. Users / Stakeholders

The AVVPIN tool has three types of users:

1. Anonymous users: Anybody may view publically accessible content on the site but cannot interact with AVPPIN functions
2. Authenticated users: Authorized bioinformaticians who are at least affiliated with Mount Sinai may create user accounts to store, visualize, and compare PPI networks, and share analyzes among members of their group(s)
3. Administrators (super-users): Site administrators will be able to manage all users and groups, content, and functions provided by the AVPPIN to users

The AVPPIN tool will serve various stakeholders, including:

1. Bioinformaticians in research labs at Mount Sinai that conduct analyses
2. Research labs at Mount Sinai that will store and serve data generated by these analyses for publication
3. General public, including the scientific community, that view and download results. They may impose additional standards on data input and retrieval from the system as a contingency on future research funding and publication opportunities.

3. Software architecture

I. Overview

The AVPPIN is a collection of the following software components that interact with each other to provide the described functions:

1. Python backend controller: core functions
2. MySQL database: data storage
3. Asynchronous task engine: analysis functions
4. Web server (e.g. Apache2): view functions
5. Web browser visualization system: interactive functions for network visualization

Additional details about each component are provided later in this section.

II. Inputs/Outputs

The AVPPIN inputs four types of data:

1. *PPI Networks:* A PPI network represents each protein as a node labeled with its name, and a relationship between a pair of proteins as an edge between their nodes. The relationships are categorical. One kind of relationship is 'interact', which is represented as an undirected edge. Other relationships are ‘up-regulates’ and ‘down-regulates’, which are represented as directed edges from the regulatory protein to its target. Additionally, each edge in the network may

have additional properties associated with it, including a 'weight' representing the strength of the interaction and a 'p-value' representing the probability of a null interaction.

Two types of networks may be used:

- Reference PPI networks for particular human tissues obtained from populations of unaffected individuals, which are called "unaffected networks".
- One or more PPI networks for the same tissue obtained from samples of subjects who have an illness -- affected individuals. These are called 'affected networks'.

Finally, the PPI networks and related information are research data that must not classify as Protected Health Information (PHI) under HIPAA to be used in this system.

Data types that can be loaded into the system (at this time) will be:

- txt files, delimited, with headers, where at least two columns are denoted as source and destination nodes (for both undirected and directed PPI networks), and an optional third column denotes the interaction type
- gml files (Graph Modeling Format), commonly implemented by many programming languages and visualization tools such as Gephi and Cytoscape
- cys files (Cytoscape JSON Format), commonly used by Cytoscape for network visualization

2. *Gene Ontologies:* This information classifies proteins as members of certain genetic pathways or biological activities, e.g. as provided by MSigDB.
3. *Gene Identities:* This information matches proteins to one or more genes, transcripts, and genomic regions across different data sets and ID systems, e.g. as provided by Ensembl.
4. *Metadata*: The tool accepts metadata about each PPI network uploaded to the system as well as the study it belongs to. The tool also accepts metadata about users and their use of the AVPPIN (e.g. user credentials and the studies and networks that belong to them). These should be delimited text files (e.g. tsv, csv).

Outputs from AVPPIN are aligned with research and data storage needs and fall into the following types:

1. *PPI Networks*: Complete or subsetted PPI networks that are either stored or generated by the system may be retrieved for further analysis or publication in the same file formats as the input file types. Annotations, such as by *GO* or *Gene Identities* may be added to the PPI network before download. Additional analyses may add additional node and edge labels into the PPI network and change its inherent structure. *Metadata* about the PPI network may be embedded into the output file.
2. *Results*: Data files produced by various analyses may be returned to the user as downloadable files that the user can retrieve. At this time, the formats of the returned files are unknown; however, they may be compressed before they are served to the user.
3. *Images*: Certain analyses may produce images from PPI networks, and will be served to the user in a lossless format such as .PNG or .PDF (with embedded SVGs).
4. *Web interface:* The tool will furnish a web interface to the user using HTML/CSS/JS rendered in a web browser such as Chrome that will allow the user to perform all of the functions as described in the Functional Overview.
5. *API:* The tool will furnish an web API which provides an abstraction layer on top of the system and allow automated software tools to interact with it. These interactions will occur via JSON files passed between end users and the service, allowing these tools to perform certain

RESTful operations on the data and perform most of the functions as described in the Functional Overview.

III. Organization and Components of AVPPIN

In a nutshell, the AVPPIN software flow will follow the State-Action-Model (SAM) design pattern: that is, users will perform functions through the view (frontend) that are executed as actions (compute engine) to update the model (database) and return the next state (results to frontend). The components of the system are shown below in Figure 1.

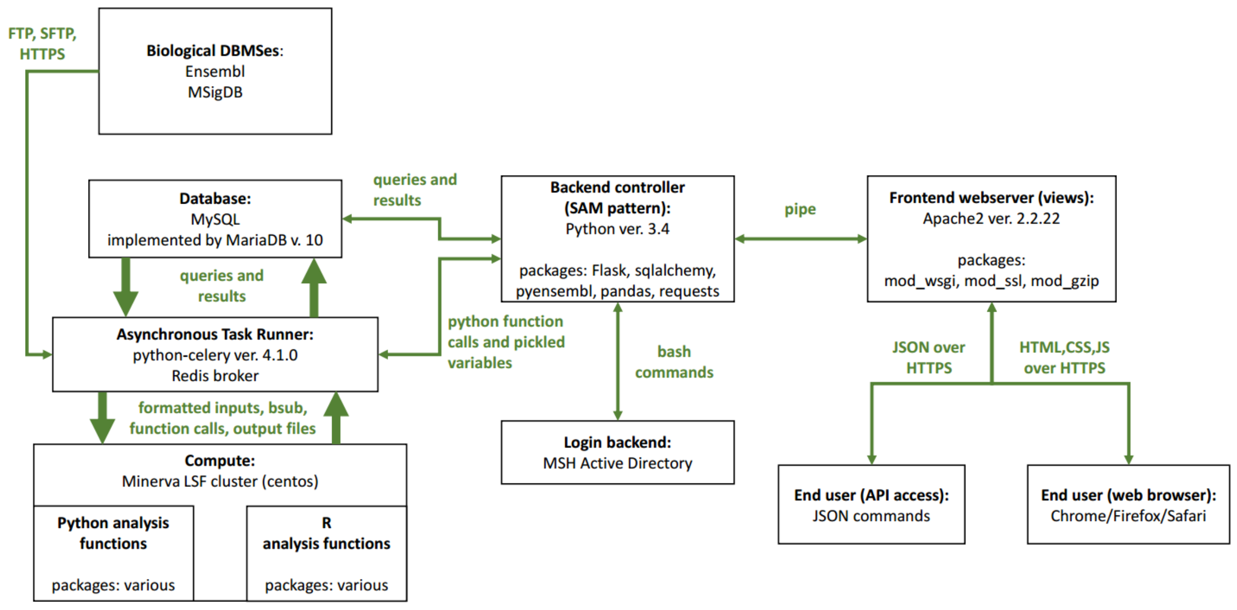
FTP, SFTP, HTTPS
Biological DBMSes: Ensembl MSigDB
Database: MySQL implemented by MariaDB v. 10
queries and results
Backend controller (SAM pattern): Python ver. 3.4
packages: Flask, sqlalchemy, pyensembl, pandas, requests
pipe
Frontend webserver (views): Apache2 ver. 2.2.22
packages: mod_wsgi, mod_ssl, mod_gzip
queries and results
Asynchronous Task Runner: python-celery ver. 4.1.0 Redis broker
python function calls and pickled variables
bash commands
JSON over HTTPS
HTML,CSS,JS over HTTPS
formatted inputs, bsub, function calls, output files
Compute: Minerva LSF cluster (centos)
Python analysis functions
packages: various
R analysis functions
packages: various
Login backend: MSH Active Directory
End user (API access): JSON commands
End user (web browser): Chrome/Firefox/Safari

Figure 1. Overview of Software Architecture Organization of AVPPIN.

As shown in Figure 1, the major components of the system are as follows:

1. Backend controller: This will be a custom-built package in Python that implements the code necessary to serve web pages (generate web pages from results/user input, provide visualization libraries, create user accounts, log in and out, provide the web API), ingest data from a variety of formats and convert it into the database model, error checking input data, output data and results, and other various core controller functions. This will also create an abstraction layer (model) on the database so that users can more easily access the data, as well as for access by other Python functions; therefore, any access to data will be modeled by the backend controller. The backend controller will also direct other parts of the system, such as the asynchronous task runner, to perform analyses on behalf of the backend controller and return results to it, to the database, or itself (e.g. in a pipeline of commands). A set of statistical functions implemented in Python and R (with their various I/O requirements) will be abstracted/wrapped by the backend controller as Python functions that can be submitted to the asynchronous task runner for computation on the compute cluster (Minerva LSF). This backend function will, at a minimum, require networkx (Python network manipulation library), Flask (web service microframework), sqlalchemy (Python SQL library), pyensembl (Ensembl Python helper library), pandas

(data manipulation), and requests (data I/O). These libraries will be reused as they are well-maintained and have become standard practice in the field; furthermore, it would be impractical to build these components ourselves to the same standards.

2. Asynchronous Task Runner*:* This is an off-the-shelf component that allows for the execution of long-running jobs off the main backend controller thread, for example, to facilitate large data analyses and backend operations, to get around the Python GIL, and to maintain responsiveness to user input while commands are running (e.g. to cancel or restart a job). In this implementation, the asynchronous task runner will retrieve data from the database in the model provided by the backend controller and return it as input to Python and R functions that will be scheduled to execute on the Minerva LSF compute engine and return results to the task runner on completion. The task runner will then deposit the results in the database for future retrieval by the backend controller. This will reuse: Celery, a Python library that creates parallel Python threads in response to Python function calls and variables passed to it through a broker; and Redis, the messaging broker that connects the backend controller to the asynchronous task runner. These were selected because of their high reliability, ease of setup, and personal experience using on previous projects (see https://github.com/ehhop/ehhapp-twilio).

3. Database: This will be a relational database (RDBMS) based on MySQL that will have a variety of tables for storage of PPI network data, user credentials and metadata, study and network metadata, and results. The database schema is given in the Database section later on in this document. A relational database was selected because of its wide support, speed with indexed network data of smaller sizes (like PPI networks), flexibility to support the other data types required, availability on the Minerva compute cluster, and standards support. Again, it would be impractical to build a custom equivalent system that meets the above criteria and performance characteristics ourselves.

4. Frontend web server: This will be a Apache2 web server that can efficiently transmit web pages and documents provided to it by the backend controller to end users, including via web browsers and the API. Apache2 is also built to facilitate rapid uploads and downloads of data, including on-the-fly compression via gzip (mod_gzip) and managing multiple instances of the backend controller to scale somewhat to demand (mod_wsgi). Apache2 also allows for HTTPS encryption (mod_ssl) of data to and from end users, which is especially important even within hospital networks due to the risk of network snooping. Again, it would be impractical to build an custom equivalent system that meets the above criteria and performance characteristics ourselves.

5. Web browser visualization system: This will be accomplished by D3.js, a network visualization library that is interactive, and/or the Plotly API for interactive plots, loaded in the client browser. These can support the network formats listed.

The AVPPIN interacts with other existing components/systems:

6. MySQL instance on Minerva: Data generated by use of the AVPPIN and user profile data will be deposited in a MySQL instance running on the Minerva supercomputing cluster. This provides the compute resources and performance required for AVPPIN.

7. Minerva LSF Queue: Long-running tasks/analyses will be sent to Minerva's LSF queuing system for batch processing, using bsub. This is the only way to access the thousands of compute cores with larger RAM availability required for advanced PPI network analyses. Functions run by the batch processing system can be any executable, but for this AVPPIN, will be Python and R functions. This will pass results back to the asynchronous task runner in whatever format is defined by the analysis performed.

8. Mount Sinai Hospital (MSH) Active Directory (AD): For the purposes of the AVPPIN, authenticated users and administrators will be active MSH affiliates with logins. These will be checked by the backend controller at every login to ensure the linked account is active.

9. Biological DBMSes: The AVPPIN requires versioned data definitions that may be loaded into the database as annotation data for PPI networks to properly match data in the database:
   - Ensembl (ensembl.org): Provides information for the classification of proteins, genes, and transcripts in many releases that change over time
   - MSigDB (software.broadinstitute.org/gsea/msigdb): Provides an annotated Gene Ontology database in many releases that change over time

   These data may be downloaded asynchronously over FTP/SFTP/HTTPS periodically or as defined by input PPI network data BioDBMS version requirements.

IV. Database

An overview of the database structure, including important indexes, foreign keys, and table relationships, is given in Figure 2 shown below.

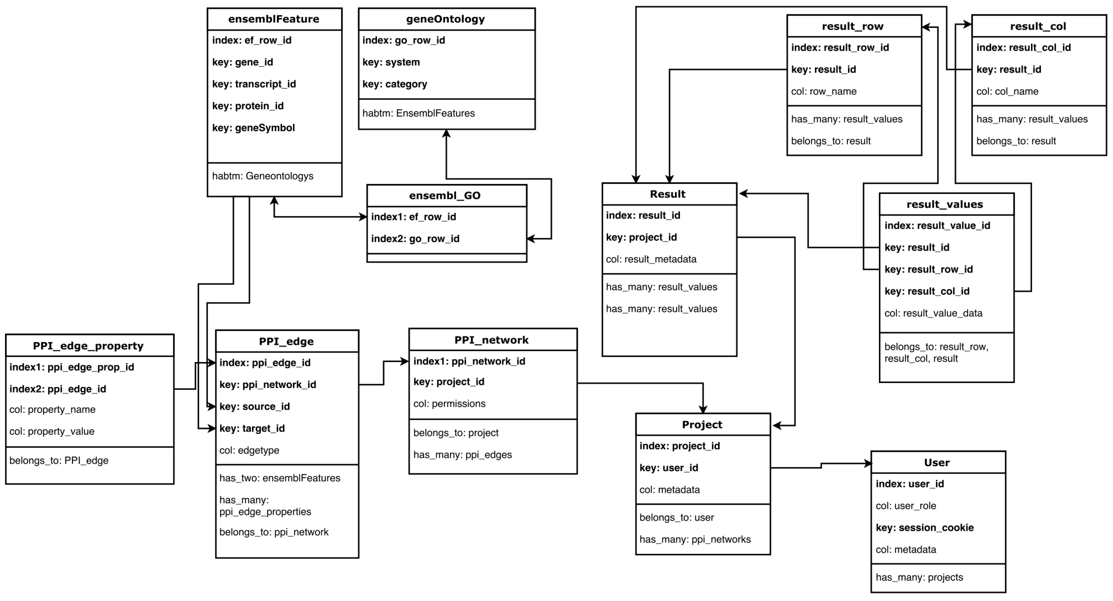

Figure 2. Database schema for AVPPIN. Note that this is not an exhaustive list of possible table column names; only the minimum necessary for AVPPIN is shown.

A brief description of each table in the database is given below:

Table Name	**Description**
ensemblFeature	Stores information about genes, transcripts, and proteins as provided by Ensembl. The layout of this table matches what is in Ensembl's DB so that it can be used by libraries like pyensembl.
geneOntology	Stores information about GO groups from MSigDB. These are groups of genes that are enriched in certain biological or chemical processes or upregulated/downregulated in response to different exposures (e.g. a treatment or a disease).
ensembl_GO	This table links ensembl features to GO groups, as each GO group represents a list of genes and each ensembl feature may belong to many GO groups.
User	Information about each user that is registered (or is an admin). Each user has many projects they are working on. Each user has a metadata BLOB associated with it which may be structured.
Project	Information about each project that is being run on AVPPIN. Each project has many PPI networks and results associated with it. Each project has a metadata BLOB associated with it which may be structured.
PPI_network	Information about each PPI_network in AVPPIN. Each PPI network has many edges between features (like proteins) as described in ensembl. A PPI network may be accessible to all or specific users. Each PPI network has a metadata BLOB associated with it which may be structured (not shown).
PPI_edge	Information about each edge in a PPI network. Each edge has a source, destination, and type (directed, undirected, upregulates, downregulates) as well as many other potential properties.
PPI_edge_property	An annotation on the edge of a PPI network. These may be anything and are stored as key:value pairs.
Result	Information about a result performed on a project, including the status of the analysis run (in the case the result is still pending, did not run, crashed, completed, has warnings, etc.) which may be stored in a metadata BLOB associated with it. Each result may be associated with tabular result data stored in the database as (row, col, value) for fast processing; higher dimensions are stored with 'value' and parsed during analysis later. Each result object should only be associated with one table of results.
Result_row	Names of the rows in a result table.
Result_col	Names of the columns in a result table.
Result_values	Values indexed by result ID, row, and column.

Loading data will be performed by custom-made Python scripts in the backend controller that convert input files into this model structure and perform error checking at runtime. These custom scripts may be sent to the asynchronous task router if they are anticipated to be long-running jobs. Returning data to the user will also be performed by custom-made Python scripts in the backend controller.

Logins will be handled by the backend controller and all required user information for this task will be stored in the database. By organizing it in this way, deleting or moving user accounts is simple as each user's projects, PPI networks, and results are inherited by the User object for simple management.

V. Security

Security is always a concern for web applications, however, at this time the security needs of AVPPIN are formally undefined; recommendations are given here for how to secure such a system. Since no PPI is stored by the system, the data and systems that run the software are not required to be encrypted at rest.

Database: One can encrypt the database with a symmetric key, such as by using the \`EncryptedType\` provided by the \`sqlalchemy_utils\` Python package, or by encrypting the entire MySQL instance for added security.

Backend/Async: Each user session may be isolated into separate Python backend processes and run on an encrypted machine.

Frontend: HTTPS encryption will be used to secure data transfer between the frontend and the end user's machine. Users will create accounts that are stored in the database and use session-based authentication with salted and hashed passwords. These passwords will be unique to AVPPIN and separate from the MSH AD. Furthermore, the system will use MSH Firewalls and may only be accessible by VPN, if desired.

Compute/HPC: Handled by Minerva HPC computing team.

4. Further Considerations for AVPPIN

As mentioned, the requirements for AVPPIN have not been finalized. Many issues still require resolution:

- Which biological DBMSes will be used as feature lists and GO terms for PPI networks.
- Performance requirements for the database and backend controller
- Statistical software that will run on AVPPIN
- Which users can be authenticated/administrators (just MSH or others?)
- Who will pay for such a system? Who will be billed for the space/compute resources required?
- Security requirements as listed above
- Which Python packages will need to be reused depending on requirements of the specification
- Whether individual data and raw data files will be uploaded to AVPPIN for processing
- Whether a Python or R client is required to facilitate end-user use of AVPPIN
- Data backups, replication, exporting the entire AVPPIN easily
- How to provide data for publication as “releases” of data stored in AVPPIN that do not change