University of Waterloo
Faculty of Engineering
Department of Electrical and Computer Engineering

Pathways Predictions: A smarter recruitment/mentorship platform

Group [redacted]

Prepared by:
Ben Chapman-Kish

Consultant: [redacted]

July 26, 2022

Context

This project is an extension of a previous FYDP project called *Pathways* that was completed by group [redacted] in 2020-2021, and this report requires the context of its detailed design document, which can be found here: [redacted]. Further information or documentation on this previous project can be provided upon request.

Abstract

The *Pathways* platform already enables developers to learn the required skills tailored to job opportunities, and can recommend resources and courses based on job descriptions, but there remains a critical barrier to its efficacy. For developers, finding the right job or the best courses for learning technical skills required on the job can be overwhelming when presented with thousands of options which may have insufficient descriptions to make informed decisions. The objective of this project is to augment the *Pathways* platform with a predictive system that recommends specific courses and job postings to users by learning the patterns behind job and course reviews. *Pathways Predictions* integrates with the existing application by offering suggestions for logged-in candidates based on which kinds of jobs and courses are regarded highly by candidates with similar profiles. As a result, the experience for all kinds of users – applicants, employers, and course mentors – will be meaningfully improved while using the platform.

Acknowledgements

I would like to thank my consultant, Dr. [redacted], for providing guidance and support throughout the process, and for her understanding of the difficulties faced by the team member in completing this project. I would also like to thank Prof. [redacted] for his advice and accommodations for the circumstances of the project.

I would also like to acknowledge that this project is an extension of a previous project, and would not exist without the work done by the members of this team. I would like to thank the group members [redacted], [redacted], [redacted], and [redacted], as well as the project's consultant [redacted], for their efforts and dedication to the original project.

I hereby confirm I have received no other help, be it material or intellectual, other than what is mentioned above, in the preparation of this project and this report.

Table of Contents

List of Figures

List of Tables

1 High-level Description

1.1 Motivation

The nature of the *Pathways* platform encourages vast selection and diversity in the jobs that may be posted and courses that may be taught in order that it may be useful for candidates. The wide variety of technologies used in the software industry is ever-growing and would require many lifetimes to learn them all. Many developers are challenged with determining what skills to focus on to adapt to particular job opportunities and how to develop such skills, and likewise employers struggle to hire candidates with relevant technical skills. *Pathways* aimed to ease this process by cross-referencing the skill set requirements for specific job positions with learning courses, but this was a very limited objective. The fundamental shortcoming of this approach is the supposition that candidates have already found a job post they're intent on applying to and will complete any relevant course to get an offer, which significantly limits the scope of the platform and its usefulness to candidates, employers, and course mentors.

Most developers seeking employment are more concerned with finding a job that they would enjoy and already have several of its required skills than they are with adapting to a particular job that would require them to learn entirely new skill sets [1], [2]. Furthermore, different people are better suited for different courses and would enjoy different skills than each other — not all minds work the same. The primary clientele for the original *Pathways* are candidates that can 1. identify a couple job postings that interest them, 2. be confident that they will actually enjoy the job and not just its description, and 3. be well-suited to learning and applying the technical skills that it requires. The greater challenge of finding jobs out of thousands that a candidate will not only enjoy, but be good at, remains unaddressed — as does the prospect of finding courses to learn skills for oneself rather than as a means to a particular job.

One of the most valuable features in the platform is entirely under-utilized: the review service that lets candidates rate any course or job that they've tried on a scale from 1 to 5 stars. The reviews are presented to applicants in chronological order without any summaries, and in the case of popular jobs and courses there may be hundreds of reviews to sift through — this compounds the already daunting challenge of finding jobs or courses that are relevant to a given applicant. Sentiment analysis is performed on the text of every review to flag outlier reviews as likely fake or exaggerated, which is useful, but this data is never revealed to users. Once the user base and the number of reviews submitted surpasses a threshold, there exists enough data to reflect rating trends between candidates and jobs/courses, it merely needs to be deciphered.

1.2 Project Objective

The objective of this project is to design a predictive system that can learn patterns between review ratings, candidates, and jobs/courses – for example, which positions and industries appeal to people based on where they live or how much experience they have – and to present suggestions to candidates based on information unique to them. This supervised learning task leverages the data already being captured in the system to practically eliminate the headache of manually sorting through job/course lists and predicting how relevant and enjoyable they will be for a given candidate. This suggestions system should improve the *Pathways* platform and expand its audience by decoupling the course recommendations from specific jobs while offering job recommendations, all of which are personalized to each individual user.

To provide an optimal suggestions service to end users, several layers of abstraction and various enhancements should be implemented. The suggestions being made should improve as new data is learned, and different types of predictive systems should be compared and dynamically selected depending on how accurate they are. The system should also be responsive and able to handle large volumes of data with minimal delay to ensure a pleasant user experience. Finally, the entire suggestions service should be fault-tolerant and modular, able to be deployed across any number of servers and resistant to interruptions in service if some of the servers fail.

1.3 Block Diagram

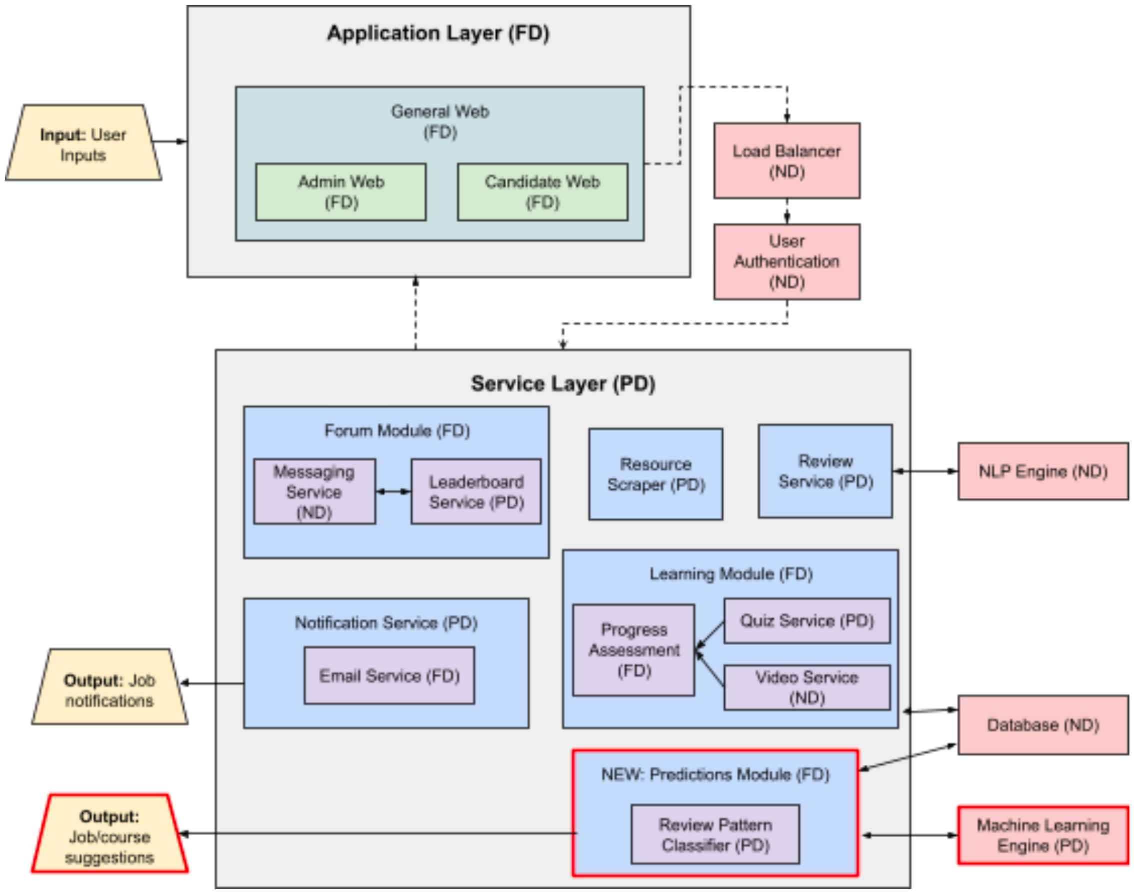

Figure 1. Block Diagram of the Pathways recruitment and mentorship platform as context (additions made by *Pathways-Predictions* highlighted in red)

The block diagram of the original Pathways platform is shown in Figure 1, with additions made for *Pathways-Predictions* highlighted in red. The detailed block diagram of the predictive system as a whole, including the four main subsystems, is shown in Figure 2.

1.3.1 Data Layer

This layer exists on the main web server and primarily retrieves and formats input/output data, interacting heavily with the main database. The learning stage is triggered on events for new reviews and changes to jobs/courses and sends labeled data to the predictions layer. The suggestions stage is triggered upon a request for job/course suggestions and sends only the applicant data to the predictions layer, receiving a complete list of estimated ratings for each valid job/course. The data layer filters and sorts these results and returns the best suggestions back to the user in the same HTTP GET request.

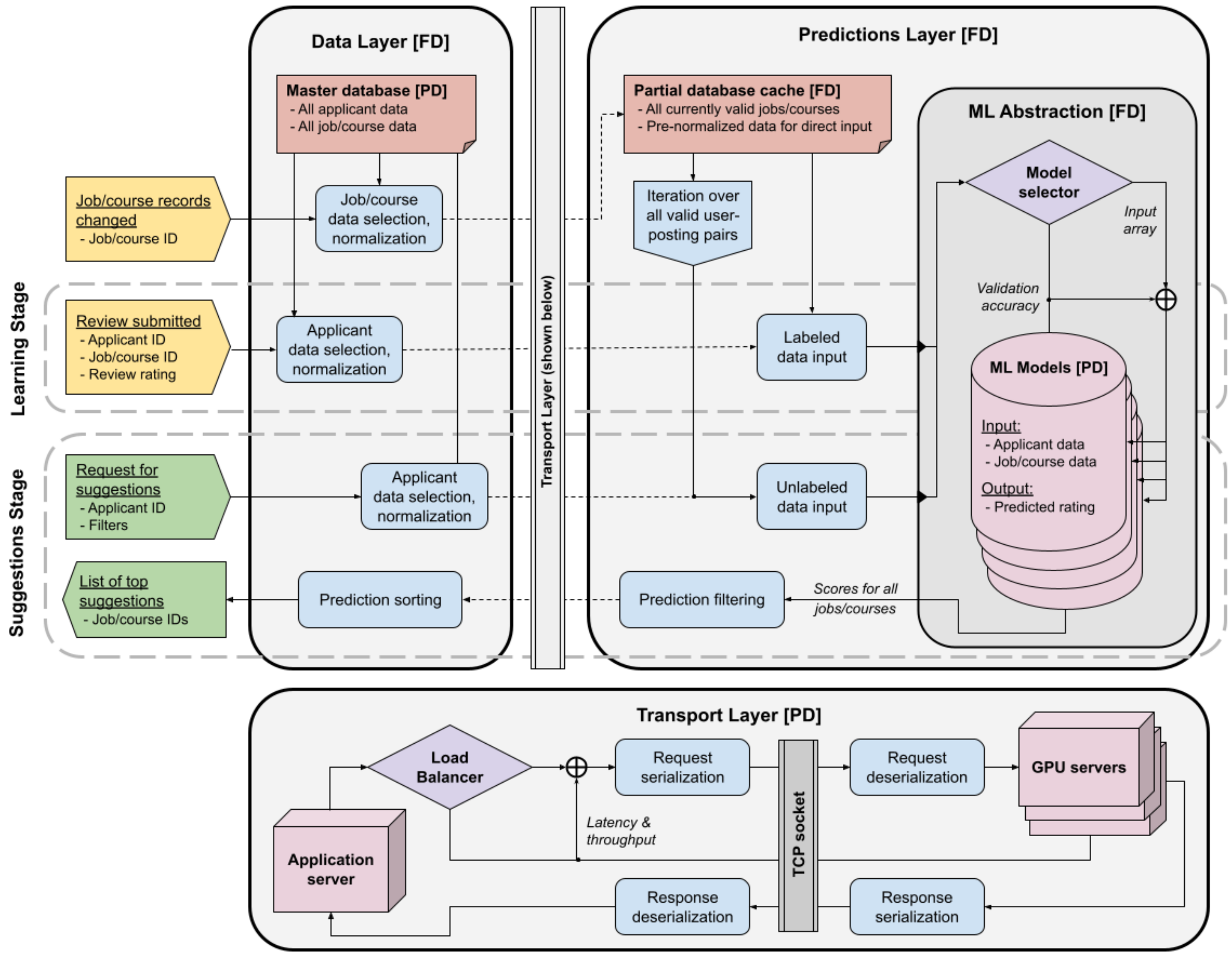

Figure 2. Detailed Block Diagram of the predictive system

Legend:

FD = Full Design	PD = Partial Design	ND = No Design	Dashed Line = Weak Association	Solid Line = Strong Association

Weak association means that the entities exchange information, but are not aware of each other. The implementation details of the participating entities are hidden, and the communication is done through an interface, as opposed to calling each other's services directly. Strong association implies that the participating entities do not appear as a black box to each other, so they can manipulate and access each other's information directly.

1.3.2 Transport Layer

This layer coordinates the communication between the data layer and the predictions layer. Involves converting data types between internal representations and JSON-serialized bytestrings for transmission over any type of Internet connection. The highly efficient data transport format minimizes time spent in transport over network links [NFS2] while allowing for interoperability between different programming languages, if desired.

This layer uses standard TCP sockets to establish a one-to-many connection between the Flask API server and any of the GPU Engine servers, allowing for dynamic host discovery and ensuring resiliency against downtime [NFS5]. Should any number of servers fail or become disconnected, so long as one or more remain functional and online, the service will continue without interruptions from the perspective of a user [NFS6].

Furthermore, the custom data transport format can handle very large amounts of data at once and includes error checks and allows for a primitive master-slave-style distributed system. Crucially, this layer also includes a custom load balancer with several server selection schemes to optimize which predictions server is being used to respond to requests based on information such as throughput and ping [NFS2, NFS4].

1.3.3 Predictions Layer

This layer is intended to be used on a machine with strong neural network capabilities — that is, powerful GPUs. It interfaces with the data layer and the machine learning models to perform the core function of making job/course suggestions to users who request it. Job and course data is received via the transport layer when updated in the main service, and unique IDs are stored in hash sets to ensure complete data replication and validity. Multi-layered caches prevent redundant data transmissions and can load pre-processed data as quickly as possible [NFS2], and when new reviews are received they are grouped into batches of raw training data for efficient training.

The learning stage, which includes all of the above, culminates in the parallel training of a variety of neural network models whose accuracies are tracked independently. The suggestions stage begins by selecting the best model based on current validation accuracy, and then pairing the applicant data up with every single valid job/course posting, feeding this already prepared data into the network as an input array. The output array represents predicted ratings for each input job/course posting, which is sent to the data layer.

1.3.3 ML Model

Each model maps applicant and job/course data to rating probabilities. Any machine learning engine or type of model can be specified at runtime, or even no intelligent model at all for a custom, although less powerful, approach. The model must implement the 3 methods of the basic model interface: learn review patterns, evaluate model accuracy, and predict ratings. Multiple ML models can be defined, and frameworks can be mixed-and-matched at will, or custom models can be written from scratch.

2 Project Specifications

2.1 Functional Specifications

Table 1. Functional specifications

#	Specification	Description	Necessity
FS1	Backward-compatibility	The system should be fully backward-compatible with the existing Pathways system: it should be possible to apply a single changelist to automatically upgrade the Pathways API server and MySQL database to plug in to the Predictions functionality without losing data or interrupting the user experience for more than 30 seconds	Essential
FS2	Submit Suggestion Requests	Candidates should be able to submit requests for suggestions for jobs or courses they may like	Essential
FS3	Validity of Suggestions	Job and course suggestions should consist of real, unexpired postings on the service that the candidate can apply to/register for (within a margin of up to 10 minutes), and every such posting should be evaluated in making suggestions	Essential
FS4	Personalization of Suggestions	Job and course suggestions should be unique to the user requesting them and be presented in order of predicted relevance	Essential
FS5	Continuous Learning	The predictions module should continuously learn from new reviews to make better suggestions (i.e. improve testing accuracy) without restarting the application or training again from scratch, with "continuously" being defined as requiring no manual intervention or use of scheduling/timers, instead happening as soon as possible	Non-essential
FS6	Model Selection	The predictions module should be able to train multiple different models and evaluate their performance, such that it can dynamically select the most accurate model when making suggestions	Essential

2.2 Non-functional Specifications

Table 2. Non-Functional specifications

#	Specification	Description	Necessity
NFS1	Model Accuracy	The predictive model used to generate suggestions should have at least 75% accuracy on real data (or demonstrate ability to fit to sample data with 80% accuracy using 5-fold cross-validation)	Essential
NFS2	Suggestion Request Latency	Suggestions must be sent to the requesting candidate within 2 seconds of receiving the request, including data preparation, model execution, suggestion filtering, and output formatting	Non-essential
NFS3	Learning Speed	The predictive model should fit to new review data within 10 seconds per 1000 reviews	Non-essential
NFS4	Scalability	The network should be able to evaluate at least 10,000 job postings or course offerings per suggestion request	Non-essential
NFS5	Resiliency	The web server should be resilient, having at least 95% uptime and protected against request overloading with the use of a load balancer	Essential
NFS6	Fault Tolerance	The backend predictive servers should be able to be started up or killed/disconnected any number of times and connect automatically to the main API server without any indication to the users, with the data remaining fully valid at all times, and all services should function as long as at least one backend server is still online	Essential

3 Detailed Design

3.1 Data Layer Subsystem Design

The data layer of *Pathways Predictions* handles all data retrieval and interfacing with the web application and the main database, ensuring that input data is complete, well-formatted, and consistent, and that output data is easy to digest. The data layer is responsible for connecting to the predictive engine servers which can be hosted on different networks, as well as selecting the fastest servers to handle requests to optimize response time and scalability [NFS2, NFS4], and it maintains reliability by monitoring backend server status and using fallback servers if a primary one becomes unavailable [NFS6]. As the original *Pathways* project was written in Python with Flask as the main web server and MySQL as the database, these selections will be preserved in this project so as to maintain complete backward-compatibility [FS1].

3.1.1 Request Handling and Triggers

The obvious input to the system are requests for job/course suggestions, which are made as HTTP requests to corresponding API endpoints listed in Table 3 in the Flask application. The format of the requests are GET with the header fields userID. The results are returned to the requesting user in the same HTTP response in a JSON list of ID-confidence pairs [FS2, FS4].

Table 3. New public API endpoints

Endpoint	Type	Header fields	Response body
/api/job_suggestions	GET	User ID	List of job suggestions (as posting IDs)
/api/course_suggestions	GET	User ID	List of course suggestions (as posting IDs)

The predictions being made by the system are on a per-posting basis, meaning that specific jobs and courses must be fed as input to determine the predicted rating that they would receive. This requires the Data Layer to also inform the Predictions Layer of any changes to the list of valid jobs and courses [FS3].

Behind the scenes, whenever a new review is published on the platform, the Data Layer will capture and process its data before sending it to the Predictions Layer for network training [FS5]. During this step, data processing and formatting can be done in the background to ensure that requests for suggestions can be handled as efficiently as possible.

3.1.2 Database Schema Extension

The database schema originally designed contains enough fields to allow for some basic pattern recognition, but could be extended to make much more personal suggestions and improve the learning capabilities of the system. The three SQL entities relevant to this are User, Job, and Course, as the Review entity merely references two of these with a rating value. We want to preserve the BCNF normalization methods to ensure good query performance, while avoiding entity structures that would be too complex to maintain or use in the data processing module [NFS2, NFS4]. Most existing data fields will be unchanged, but fields which are foregin keys on other tables with hardcoded values can be replaced with enums.

Table 4. User entity

Field	**Type**	**Description**
user_id	char	Unique user id for the user
region	enum	Current region
industry	enum	Industry related to the user
education_level	enum	Highest level of education achieved
experience	int	Years of relevant professional experience
average_grade	decimal(3,2)	
tags	array[bool]	Skill tags selected/not selected for the user

There are no foreign keys in this table, and every attribute is functionally dependent solely on the primary key, which satisfies the BCNF requirements.

Table 5. Job entity

Field	**Type**	**Description**
job_id	char	Unique job id for the job posting
region	enum	Region of the job
industry	enum	Industry related to the job
position_level	enum	Seniority level of the position
company_size	int	Number of employees at the company
salary	int	Estimated salary
tags	array[bool]	Skill tags selected/not selected for the job posting

There are no foreign keys in this table, and every attribute is functionally dependent solely on the primary key, which satisfies the BCNF requirements.

Tables 4, 5, and 6 show the new fields in the entity structure design, including relevant or transformed existing fields in italics. Note that the tags field is represented as an array here, but this violates 1NF and would actually be expanded as several bool fields in practice.

Table 6. Course entity

Field	Type	Description
course_id	char	Unique course id for the course offering
industry	enum	Industry related to the course
num_videos	int	Number of videos attached to the course
tags	array[bool]	Skill tags selected/not selected for the course posting

There are no foreign keys in this table, and every attribute is functionally dependent solely on the primary key, which satisfies the BCNF requirements.

Finally, we can find the average grade for either all users in a course, or all courses for a user, by performing AVG operations on Table 7:

Table 7. Grade entity

Field	Type	Description
course_id	char	Unique course id for the course offering
user_id	char	Unique user id for the user
grade	decimal(3,2)	Grade received in the course for the given user

The primary key is {course_id, user_id} and the grade attribute is functionally dependent solely on this key, which satisfies the BCNF requirements.

3.1.3 Sample Data Generation

Although this project is designed to be used by real people in real time, with an ever-growing database of people's reviews of actual jobs and courses they've tried, this is simply not possible for a fourth-year design project. Instead, sample data is generated with a developer API endpoint that can create thousands of fake users, job postings, and courses with randomized data. But for the purposes of training machine learning models, the way fake reviews are generated has to contain a realistic, somewhat complex pattern behind them. To achieve this end, a probabilistic approach has been used to simulate people's preferences for jobs or courses based on their own personal details, and a complex multi-weight rating simulation has been designed to create fake reviews that, with some margin for personal preference or emotional ratings caused by factors invisible to the types of data we collect, should paint a decent picture of what types of users will enjoy certain types of jobs or courses.

3.2 Transport Layer Subsystem Design

The initial design of this project involved having all the machine learning on the same machine as the API server, which was a much simpler design. This obviously had numerous disadvantages: it offered no ability to parallelize tasks, required the API server to be run on a powerful GPU-enabled machine, resulted in an extremely slow user experience with most of the platform's webpages taking minutes to load, and this resulted in a single point of failure should any machine learning model break. However, the only way around this issue is to design a way to somehow connect the API server to one or more machine learning servers over the internet.

The transport layer facilitates this in a sort of master-slave system, allowing the main API server to maintain complete control of data replication and consistency, while also providing a very efficient mechanism by which these servers can communicate with each other. The core of this layer is built on basic web sockets operating on the IP transport layer. An engineering design trade-off has been made between speed and size of communication, preferring to minimize latency of the primary service even if it means sending more bytes than strictly necessary.

3.2.1 Serialization Format

The transport layer needs a serialization format with which to encode and decode data that can be transmitted as a byte stream. Proper selection of this format is essential for ensuring no requests are lost, maintaining data integrity and low latency, and supporting large scales of data [FS2, FS3, NFS2, NFS4]. There exist a variety of options that can fulfill this role, the most widely-used ones being XML, JSON, ProtoBuf, and Pickle.

The most important aspects of a serialization format to compare relate to its speed, compression factor, and support for various use cases. To compare these attributes, qualitative assessments and quantitative benchmark results generated by Shmuel Amar [3] are examined for the following categories:

- Serialization speed: how long the library takes to serialize and deserialize 1 million objects (μs).
- Compression factor: the size of the serialized form of 1 million objects when using the given library (bytes).
- Language support: the amount of mainstream programming languages that have official support for this serialization format, graded from worst to best as {only Python, some, most}.
- Human readability: Whether or not this data format can be read by humans, which helps with debugging (either Yes or No).

These results are summarized in Table 8:

Table 8. Performance and compatibility assessments for serialization formats [3]

Format	**Average encoding time (µs)**	**Object compression size (bytes)**	**Language support**	**Human readable**
XML	12.44	1,060.78	Most	Yes
JSON	8.00	781.82	Most	Yes
ProtoBuf	48.66	299.46	Some	No
Pickle	3.76	389.5	Only Python	No

These formats are compared using a weighted decision matrix. The criteria/weights are:

- Encoding speed (40%) – The average encoding time from Table 8. For this criterion, values on lower orders of magnitude are better so the square inverse of each value is used as input.
- Compression (25%) – The object compression size from Table 8. For this criterion, lower values are better so the inverse of each value is used as input.
- Language support (20%) – A numeric transformation of the value from Table 8, in which 'only Python'=1, 'some'=3, and 'most'=5.
- Human readability (15%) – A boolean corresponding to the value from Table 8, in which 'no'=1 and 'yes'=3.

The weights and numeric transformations have been selected based on importance to the successful completion of the project and fulfillment of its specifications. The resulting decision matrix is shown in Table 9.

Table 9. Weighted decision matrix for transport layer serialization format

Format	**Criteria**								**Total**
	Encoding speed		Compression		Language support		Human readability		
	Raw	Norm	Raw	Norm	Raw	Norm	Raw	Norm	
XML	√(1/12.44)	0.550	1/1060.78	0.282	5	1	3	1	**64.1**
JSON	√(1/8.00)	0.686	1/781.82	0.383	5	1	3	1	**72.0**
ProtoBuf	√(1/48.66)	0.278	1/299.46	1	3	0.6	1	0.333	**53.1**
Pickle	√(1/3.76)	1	1/389.5	0.769	1	0.2	1	0.333	**68.2**

From the results obtained in Table 9, JSON scored as the best option for the serialization format to use in the transport layer of the application. Its serialization speed is impressive and the lower compression factor is made up for by its human readability and widespread language support, so it should function well for its purpose.

In the final design, the predictions layer connects to the main web application using basic web sockets operating on the IP transport layer. Incoming and outgoing data is serialized with speed as a priority rather than transmission size. The predictions layer remains in an idle event loop until a socket transmission is received, at which point it spawns a thread to either train the corresponding models or use them to make predictions. When a request reaches this stage, it includes all the normalized features for the applicant in question but it must pair this data up with every possible job or course in the local cache. The top ten prediction results in the form of a list of job/course IDs are returned to the data layer when the request is fulfilled.

3.2.2 Data Exchange Protocol

A custom protocol has been written for the specialized purposes of this application. Using the selected serialization format, messages will be encoded as either *requests* to a predictions server, or as *responses* back to the main API server. All responses indicate whether the request was handled successfully or not, as well as the reason why not if a failure happened. All possible protocol formats are specified in Table 10.

Table 10. Transport layer request formats

***RequestType* enum**	**Description**	**Supplied data**	**Expected response**
PING	Used to measure network latency and server overhead	None	None
DISCONNECT	Used to indicate the API server is shutting down	None	None
POSTINGS_CHUNK	A batch of postings to be cached on each backend server	List of postings	None
POSTINGS_DONE	Used to indicate a set of postings is done being transferred	Total postings transferred	None
REVIEWS_CHUNK	A batch of reviews to be cached and trained with on each backend server	List of reviews	None
REVIEWS_DONE	Used to indicate a set of reviews is done being transferred	Total reviews transferred	None
SUGGESTIONS	The core feature of requesting suggestions for jobs or courses that a specific user may like	Requesting user, requested posting type (job or course)	List of suggested posting IDs

3.2.3 Load Balancer Design

The Pathways Predictions service is designed as a master-slave distributed system between the single API server and any number of backend machine learning servers. This was arrived at after an iterative design process that first began with a single ML server that had to be linked in advance to the API server. This grew to also contain a list of backup servers that can be switched to should the original one fail (albeit with a lot of overhead – the whole server had to start up again), and following that a system was designed to allow for training ML models on all of the backup servers at once. The final design extends this even further by employing a load balancer to track the loads and network delays of each of the backend servers.

The load balancer is responsible for maintaining a smooth user experience by balancing requests across all of the backend servers. It keeps track of both latency and rolling-average throughput for all online servers while distributing suggestion requests across them all so that each server experiences a roughly equal load [NFS5]. When sending only data to the backend predictive servers, the load balancer will ensure data consistency and give each server the same information, verifying that it has been received.

When a request for suggestions is made, the load balancer selects only one backend server to handle the request as it doesn't change any data, and the results are presented to the user without implementation transparency. The load balancer has several schemes for selecting which backend server to use, which are:

- No method (always select the first available server in the active server list)
- Random server
- Cycle through servers
- Lowest ping (based on the most recent request made to the server)
- Highest throughput (based on a 10-request rolling average of response times made for the particular request type in question)

At server startup, the choice of this scheme can be configured in the configuration file.

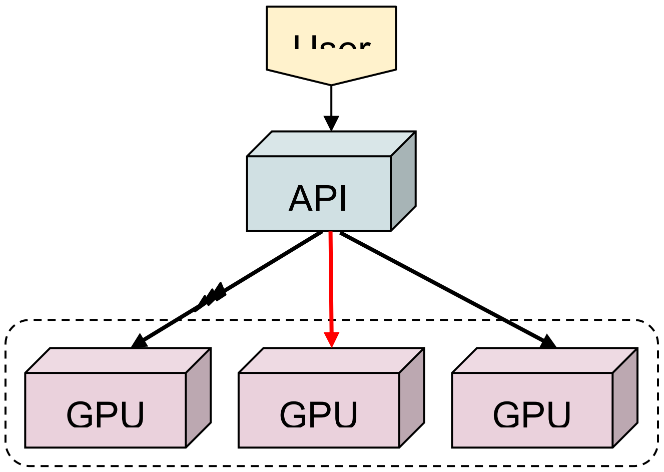

Figure 3. A typical configuration of the load balancer

In Figure 3, a possible configuration of the one-to-many relationship between the API server with the load balancer and the GPU servers. In this illustration, GPU server 1 is supposed to be the next server to handle a user request, but it is disconnected from the API server (denoted by the lightning bolt). When the load balancer tries to dispatch the request, it will detect this and dispatch the request to GPU server 2 instead. Should that server fail as well, the request will be dispatched to server 3, and if that one also fails then there are no remaining GPU servers to handle the request and the API server will notify the user of a server failure.

3.2.3.1 Fault Tolerance

If a server fails or a connection is lost, the load balancer removes it from the active server list and it will have to be re-discovered in the future. When a server is re-discovered, its local caches are compared against the master database and any missing data records are sent over again, which reduces overhead on the server.

A scheduled check every 10 minutes verifies the online status of every server in the server list, removing any that have failed, adding any that are now available, and ensuring that every server has a complete and valid list of postings and reviews. Predictive servers can become temporarily disconnected from the main API server, or only become online after the API server has already been started, and any infinite sequence of these events should still result in a complete and functional operation [NFS6]. In manual testing, servers have been killed or disconnected from the internet hundreds of times over, and they continue to always function as expected, without ever affecting the user who is trying to use the platform.

3.3 Predictions Layer Subsystem Design

The predictions layer of *Pathways Predictions* handles all the heavy lifting of actually learning patterns behind reviews and making predictions based on these insights. By design, this layer can be operating on an entirely different machine than the main API application; ideally one with powerful GPUs so that the machine learning library can parallelize the process further and minimize latency and training speed [NFS2, NFS3]. This distributed nature of the service not only maintains scalability [NFS4], but adds a level of fault tolerance in case one system is overloaded [NFS6]. Theoretically, any number of machine learning servers could be added to the system but considering the web framework is centralized and single-threaded, this would not be the main bottleneck of the system.

3.3.1 Interface with Abstract Models

All operations involving training on new data, determining accuracy, and making predictions are done through an abstract model interface that can be implemented by virtually any system whatsoever — it need not be from a well-known machine learning framework. Custom implementations can be written or multiple models with different parameters can all be defined, so long as they implement the three required methods of this stage. The data format is in Table 11:

Table 11. Features used by the model selector for training

Applicant data	Industry (enum)	Region (enum)	Education level (enum)	Experience in years (int)	Average quiz grade (float)	List of tags (bool x8)
Job data	Industry (enum)	Region (enum)	Position seniority (enum)	Company size (int)	Estimated salary (int)	List of tags (bool x8)
Course data	Industry (enum)	Number of videos (int)	Average quiz grade (float)	List of tags (bool x8)		

In classification systems this data is paired with review ratings as a discrete list of enumerated types, but in regression the ratings are regarded as floats.

3.3.2 Automatic Model Selection

The original design for this project involved finding the single best neural network to make suggestions on the platform, which would have been a lot less work, but the caveat of this approach is that our service would be mostly unable to adapt to new data. Should any trends in some industries change, or if salaries become more important after a recession, or any number of factors, we would be stuck using a model that is no longer the best-suited for the task. Even aside from this, we would need to tune our hyperparameters expertly and have unrealistic confidence that the best network possible has been constructed. Instead, a model selector is used to train a variety of machine learning models at once and dynamically select the most accurate one for each request it receives.

As each machine learning model is trained on new review patterns, their validation accuracy data is reported to the model selector. Because the learning stage is functionally separate from the suggestions stage, this adds no latency to the generation of predictions as it can use whichever model is not being actively trained. When multiple models are available as a suggestions request comes in, the model selector automatically selects the best model at that point in time. If one model is better suited for course suggestions than job suggestions, this is taken into consideration as well.

3.3.3 Feature Selection and Data Normalization

An important process of machine learning is choosing the very data that is going to be trained on — the numeric components of a piece of data which are called features. Not enough relevant features results in overly general models that can't learn very complex patterns, but having too many irrelevant features will result in models that overfit to patterns which aren't representative of the real problem being captured. It's important to select only the most appropriate features when training a model, which in this case are the exact fields of the database schemas defined above, conveniently transformed into types that can be effectively used in machine learning. The features used for this operation are listed above in Table 11.

3.4 Machine Learning Models Subsystem Design

The machine learning models serve as the backbone of the entire project; without them, there would be no *Pathways-Predictions* service. Machine learning is a complicated field, and a moderate knowledge of its principles is a prerequisite to understanding this section of this report.

3.4.1 Machine Learning Frameworks

The machine learning frameworks are integral to the operation and development of the suggestions stage. The main purpose is to facilitate the generation of relevant job/course suggestions for candidates. To do this, the framework has four primary responsibilities:

1. Learn review rating patterns on-line [FS3]
2. Predict ratings of specific job/course posts for specific users [FS4]
3. Achieve a high degree of accuracy with predictions [NFS1]
4. Be able to process large degrees of data in short time periods [NFS2, NFS3, NFS4]

Any serious machine learning framework should be able to satisfy these requirements, but their differences lie in metrics such as latency, parallelization, performance, ease of development, and other such qualities. Among the machine learning frameworks in existence, the ones to consider are a small selection of mature and popular alternatives which are all written in Python, including TensorFlow/Keras, PyTorch, PyCaffe, and Theano.

Each of these frameworks has considerable support from software communities and are used pervasively in a variety of applications, however there is some functional difference in how their machine learning models work. Although several frameworks can be employed at once, only those which are best suited for the application, judged with quantitative technical assessments and qualitative evaluations, will be selected.

3.4.1.1 Framework Performance

One of the most significant aspects of a machine learning framework to consider is its performance — that is, how fast it runs and how well it trains. To compare performance, benchmark data published by a group of deep learning researchers [4] can be used. The metrics to be examined are:

- Training time: how long the framework takes on average to train a model on the MNIST dataset (s).
- Testing time: how long the framework takes on average to test a model with the MNIST dataset (s).
- Accuracy: how well the framework can train a model on the CIFAR-10 dataset (%)

Table 12. Benchmark data for selected frameworks using CUDA GPUs [4]

Framework	**Training time (s)**	**Testing time (s)**	**Accuracy (%)**
TensorFlow/Keras	68.51	0.26	87.00
PyCaffe	97.02	0.55	75.52
PyTorch	338.46	1.73	65.96
Theano	560.04	0.19	54.49

3.4.1.2 Framework Selection

These four frameworks shall be compared using a weighted decision matrix. The criteria will match the data in Table 12, and the criteria corresponding to time values will be inverted since lower is better. The weights are being selected based on importance to the successful completion of the project and fulfillment of its specifications – training time being the least important criterion should be weighed at 30%, and testing time and accuracy will each be worth 35%. The resulting decision matrix is shown in Table 13.

Table 13. Weighted decision matrix for machine learning framework selection

Framework	**Criteria**						**Total**
	Training time (30%)		Testing time (35%)		Accuracy (35%)		
	Raw	Norm	Raw	Norm	Raw	Norm	
TensorFlow/Keras	1/68.51	1	1/0.26	0.731	0.870	1	**90.6**
PyTorch	1/97.02	0.706	1/0.55	0.345	0.755	0.868	**59.7**
SciKit-Learn	1/338.46	0.202	1/1.73	0.110	0.660	0.759	**36.5**
Theano	1/560.04	0.122	1/0.19	1	0.545	0.626	**60.6**

The results in Table 13 show that the optimal framework to use for similar classification tasks on similar machines (Linux servers with thousands of CUDA cores) is TensorFlow/Keras. This is especially convenient as this framework is the one the team is most familiar with, and so all the multi-layer models will be implemented with this framework. None of these frameworks are well suited for implementing support vector machines, however, so the SciKit-Learn library will be leveraged as it is a barebones framework but one that is entirely appropriate for the task.

3.4.2 Model Design

A machine learning network can take on many forms, but the most well-known variety is a feed-forward artificial neural network. The specifics of how machine learning works will be skipped here as that could take a whole 35 pages on its own, but the basic concept that will be leveraged by our models is the ability to fit to new data and learn nonlinear patterns. A typical deep learning ANN might look like this:

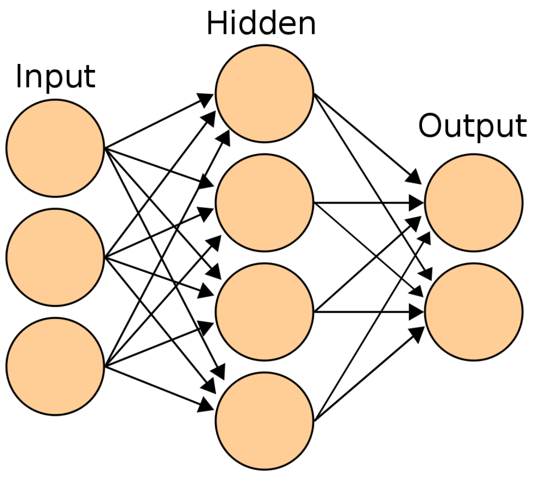

Figure 4. A feed-forward ANN with a single hidden layer [5]

Figure 4 depicts a neural network with a single hidden layer consisting of four nodes, and an output layer of two nodes, both of which are fully-connected to the layer before them. When training this model, we use labeled data to distinguish between what the output should be and what we predicted it to be. In the case of this project, the input nodes are all the relevant features for a job or course review, the output nodes are the likelihood of a given star rating being submitted for this review, and the labels are what the actual star rating was for such a review.

The number of hidden layers to use in a network, the number of nodes for each layer, the types of layers and activation functions used, and the rate at which the network fits to new data are all hyperparameters that can be adjusted depending on the need of the task.

3.4.2.1 Model Selection

Many assortments of supervised models have been tested, including:

- Fully-connected layers with sigmoid activation functions
- Convolution and pooling layers with ReLU activation functions
- Normalization and regularization layers
- Softmax activation function to predict given star ratings - classifier approach
- Linear activation function to predict the rating value directly - regression approach
- Support vector machines

The prototype data available in Section 4 indicates that the best models are implemented with TensorFlow/Keras as sequential layers of fully-connected layers. The input should be normalized in batches and dropout layers should be put between some of the fully-connected layers for regularization, which prevents overfitting and minimizes training time. For all fully-connected layers, the sigmoid activation function is the most appropriate, as tanh performed around 8% worse on average.

The loss function for classifiers was originally set to the default categorical cross-entropy loss, but eventually label smoothing with a parameter of 0.1 was adopted as it resulted in even higher accuracies. The loss function for regressors is root mean squared error, which can be converted into the exact same accuracy metric already used by the classifiers. Based on the available data, the model selector for each backend system should be configured with any of the default models listed in Table 14.

These models have been selected as they all have validation accuracies of above 80% [NFS1] and training times of less than 6 seconds per 1000 reviews [NFS3], and they encompass a wide range of model types. As such, the model selector should be able to adapt to new patterns that may emerge if the sample data used so far was too limited. As seen in Section 4, with 10,000 postings in the database, the Keras models also are all able to generate job/course suggestions in less than 0.5 seconds, and even the SVM models don't take longer than 1.5 seconds [NFS2, NFS4], so we can be confident that users won't be agitated by waiting for a neural network to take ages to generate suggestions for them.

Table 14. Best-performing models selected for the suggestion task

Model type	**Keras parameters**				**Validation accuracy**	**Training time (seconds per 1000 inputs)**
	Learning rate	**# of layers**	**# of nodes per layer**	**Dropout rate**		
Classifier	0.005	2	30	0.2	86.07%	5.24
Classifier	0.010	3	30	0.2	86.07%	4.25
Classifier	0.005	4	10	0.2	86.70%	4.84
Classifier	0.005	4	50	0.1	87.86%	5.32
Regressor	0.005	2	30	0.1	86.59%	5.23
Regressor	0.005	2	40	0.1	86.06%	3.89
Regressor	0.005	3	50	0	87.27%	4.08
Regressor	0.005	4	50	0.1	86.49%	3.87
	SVM parameters					
	Kernel	**C parameter**	**gamma parameter**			
Classifier	RBF	1	100		81.75%	1.62
Regressor	RBF	20	50		83.90%	1.66

3.4.2.2 Model Training and Performance Metrics

Training a machine learning model is challenging for several reasons – clearly, the model might not learn any of the patterns that are present in the training data, but the more sinister fate is when the model learns the training data so well that it can't generalize to new data any more, and fails to make accurate predictions when presented with unseen data. The concepts of bias and variance are essential to this, as well as the difference between training, validation, and testing error, but there isn't enough time or space to explain these concepts here.

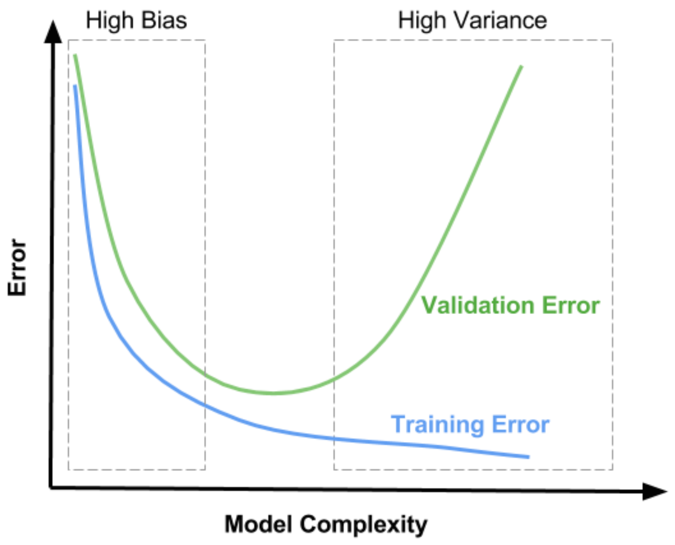

Figure 5. The bias vs. variance trade-off [6]

In Figure 5, this challenge is shown as an error curve in relation to model complexity. When a model is too simple or hasn't been trained on enough data, it has underfit to the data, which results in high bias and high variance as well as high error while training. However, if a model is too complex or has been training on similar data for too long, it starts to overfit to the data, which is harder to detect because it's associated with a low training error. The appropriate level of model complexity is an engineering design trade-off perfectly suited for a fourth-year design project, and that's exactly what we've done.

To prevent overfitting, some of the training data is separated out into a set of validation data. This data is never seen by the neural network while training, but is used to assess if more training will result in a meaningful improvement in performance. For the Keras models in this design, a technique called K-fold cross-validation is implemented with K=5 (which corresponds to a training-validation split of 0.2) and this is accompanied by an early stopping criteria of 30 epochs which will stop training the model when the validation error climbs again, restoring it to the best weights – this corresponds to the valley in the green curve in Figure 5.

4 Prototype Data

4.1 Service Throughput

With a functioning prototype that can receive and respond to requests for job/course suggestions by users of the platform, tests were conducted for the common use cases to determine how fast the service is at various scales of data. To ensure consistency, all of the following tests have been conducted with the same configuration: the frontend API server is running on a five-year-old MacBook located 100km away in Toronto, with a high-speed wireless Internet connection. There is one back-end predictive server running on *ecetesla1*, a high-end linux machine on the University of Waterloo network with 5,888 CUDA cores.

The model being used by the predictive service will be pre-determined, and each test will be repeated with each of the following models as both classifiers and regressors . Classifiers use categorical cross-entropy loss with a softmax activation function for the output, and regressors use mean-squared-error loss with a linear activation function for the output.

<u>Model A</u>: A Keras model with a learning rate of 0.01
- Two fully-connected layers with 10 nodes each and sigmoid activation functions

<u>Model B</u>: A Keras model with a learning rate of 0.01
- A batch normalization layer
- Three fully-connected layers with 40 nodes each and sigmoid activation functions

<u>Model C</u>: A Keras model with a learning rate of 0.01
- A batch normalization layer
- One fully-connected layers with 40 nodes and a sigmoid activation function
- A dropout layer with a rate of 0.1
- One fully-connected layers with 40 nodes and a sigmoid activation function
- A dropout layer with a rate of 0.1
- One fully-connected layers with 40 nodes and a sigmoid activation function

<u>Model D</u>: A Keras model with a learning rate of 0.01
- A batch normalization layer
- Two fully-connected layers with 40 nodes each and sigmoid activation functions
- A dropout layer with a rate of 0.2
- Two fully-connected layers with 40 nodes each and sigmoid activation functions

<u>Model E</u>: A SVM model with an RBF kernel, C=10, and gamma=10.

<u>Model F</u>: A SVM model with an RBF kernel, C=50, and gamma=100.

4.1.1 Model Training Time

All network times are measured directly from the Flask application on the frontend API server, and all model training times are measured directly by the model selector in the backend server, both using the *timeit* library. Each of the 12 models has been tested with 4 different amounts of reviews (from 1000 to 4000 in even increments), but for the sake of brevity only alternating models will earn rows. Each regressor has very similar performance to its classifier counterpart, so those will be omitted as well.

Table 15. Model training times

Model	Number of reviews	Validation accuracy	Training time
Classifier A	1000	77.14%	4.64 s
Classifier A	2000	78.75%	6.77 s
Classifier A	3000	78.97%	16.08 s
Classifier A	4000	79.10%	16.11 s
Classifier C	1000	84.73%	4.08 s
Classifier C	2000	84.90%	6.29 s
Classifier C	3000	85.49%	8.33 s
Classifier C	4000	86.17%	11.24 s
Classifier E	1000	79.45%	1.66 s
Classifier E	2000	80.33%	6.64 s
Classifier E	3000	80.16%	15.42 s
Classifier E	4000	80.37%	28.55 s

From the results in Table 15, it's clear that after the first 1000 reviews, we have very diminishing returns on validation accuracy. The training data seems to be simple enough to be learned without too many data points, and our early fitting criteria in coordination with K-fold cross-validation has ensured that we don't overfit to our data, which would be visible as a decreasing validation accuracy. Furthermore, training time increases differently depending on the architecture of each model.

Model A has only two fully-connected layers of 10 hidden nodes each, but the training time jumps up between 2000 and 3000 reviews. However, due to the early stopping criteria, it takes virtually no longer to train on 4000 reviews, as the validation accuracy begins to decrease around the 3000 review mark.

Model C performs significantly better than model A, both in terms of a higher accuracy achieved and much lower training times. Both of these qualities can be attributed to the both the normalization layer at the input and the dropout layers in the middle, which help prevent the backpropagation algorithm from getting stuck in local maxima, resulting in a greater peak validation accuracy which is achieved in fewer epochs.

Model E has the best training time by a large margin for the case of 1000 reviews, but scales terribly once 3000 reviews are trained at a time. Presumably, the support vector machine scales exponentially with the number of reviews used, as there is no early stopping criteria available to prevent overfitting. Since its accuracy doesn't increase after the first 1000 reviews anyway, there may be potential to achieve extremely fast training times as long as input is kept in small batches.

The conclusion drawn from these results is that batch normalization and dropout should both be employed in a Keras model to ensure regularization and achieve optimal validation accuracy while keeping the training times relatively low.

4.1.2 Model Prediction Time

Using the same 12 models as defined above, the other area of interest to investigate regarding the prototype's performance is prediction time. After training each model in the way described above, models were used to generate job suggestions for random users on the platform and various measurements were taken. For these tests, the models were trained on approximately 2000 reviews and were given 10,000 job posts to store in their cache, and each post was fed through each model to generate the top 10 suggestions for each user. The average of these tests are recorded in Table 16. Once again, only alternating models will be displayed to maintain readability.

Table 16. Model prediction times

Model	**Cache dump time**	**Data concat time**	**Pred gen time**	**Pred sort time**	**Client round trip time**
Classifier A	2 ms	14 ms	199 ms	29 ms	271 ms
Classifier C	1 ms	12 ms	208 ms	34 ms	283 ms
Classifier E	1 ms	13 ms	928 ms	16 ms	1,175 ms
Regressor A	1 ms	15 ms	180 ms	19 ms	242 ms
Regressor C	1 ms	12 ms	202 ms	62 ms	308 ms
Regressor E	1 ms	15 ms	910 ms	9 ms	1,225 ms

The column *cache dump time* corresponds to the amount of time needed to retrieve all 10,000 postings from the server's local cache. The column *data concat time* corresponds to the amount of time used to concatenate the features of each of these postings with the given user's features. The column *pred gen time* indicates how long the model spent actually processing the input and making its predictions for each job. The column *pred sort time* corresponds to the amount of time used to sort the output predictions by their score and extract the top 10. Finally, the column *client round trip time* indicates how much time elapsed between the API client making the request to the given server, and receiving the response as a list of job suggestions. Consider that this includes all the overhead regarding data serialization/deserialization, wireless latency between the client and the WiFi router, and transit time over the wide area network between the two machines.

These results in Table 16 demonstrate that for the Keras models, the bottleneck in generating suggestions for users is clearly the step where the machine learning model makes predictions on the given input. The cache dump and data concatenation are insignificant in comparison, and the predictions are sorted with an efficient bisect sorting algorithm that operates in $O(n \cdot log(n))$ time, which is also relatively minor in comparison to the machine learning step. The round-trip time for each of the Keras models, even when evaluating 10,000 samples in the machine learning model, is still less than half of a second from the perspective of a client in a different city. Noticeably, the SVM models are far slower at making predictions at this scale of data than the Keras models by a factor of around 5 times. Despite this, even they still take less than 1.5 seconds from the client's perspective. These results are even better than expected, and are a testament to the level of optimization that went into this process.

4.2 Model Performance

Users obviously would prefer to receive their job/course suggestions quickly, but speed is moot if the results are unreliable. The 12 models used for benchmarking in the previous section represent different kinds of neural network architectures, but there are an infinite amount of possibilities for how to structure a neural network, and we only want to give each model selector a small, manageable number of models to choose from in making suggestions for users. In this section, 720 different configurations of Keras models and 84 different configurations of SVM models have been constructed, including both classifier and regressor forms.

The parameters that were varied are displayed in Table 17. Every permutation of these possible values was tested, but this is too much data to display in a table, so trends can be observed with graphs instead. Figures 6, 7, and 8 show some of these trends, but the remaining 3 figures had to be cut to maintain an acceptable page count for this report.

Table 17. Values of parameters used in model comparison

Parameter	**Values**				
Model type	Classifier	Regressor			
Learning rate	0.005	0.01	0.05	0.1	
Number of fully-connected layers	2	3	4		
Number of hidden nodes per layer	10	20	30	40	50
Dropout rate	0 (none)	0.1	0.2		
Normalize input	True	False			

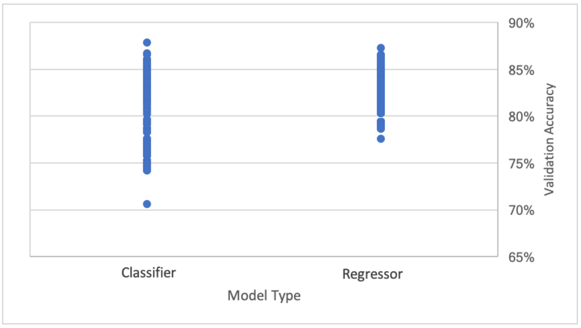

Figure 6. Model Type vs. Accuracy

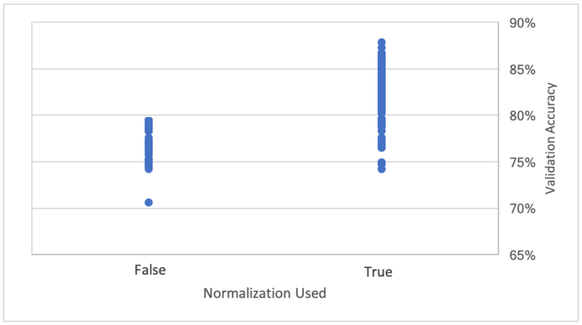

Figure 7. Input Normalization vs. Accuracy

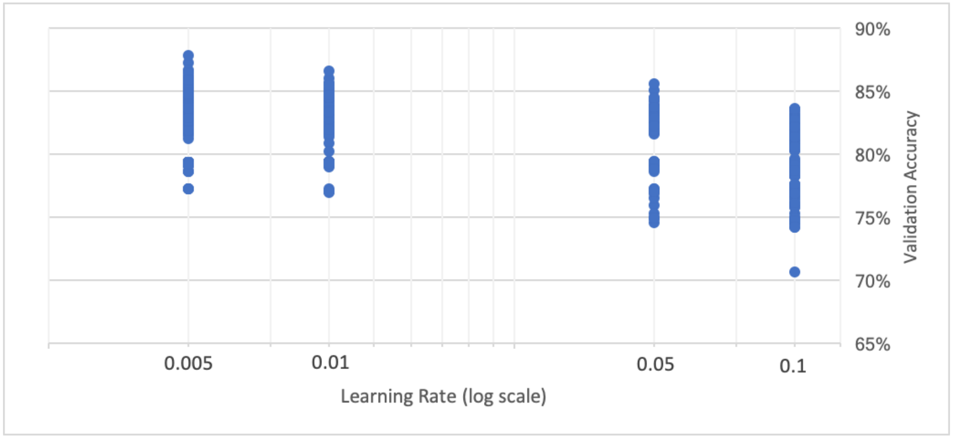

Figure 8. Learning Rate vs. Accuracy

Unfortunately, due to limitations of creating graphs in Excel, we cannot glean quite as many complex trends from these graphs as there actually are. Even still, we can observe that regressors are more consistently accurate than classifiers, and normalizing our input as well as using lower learning rates both make a significant difference in the average validation accuracy. From the raw data obtained, the models with the best accuracy all use input normalization, and there happen to be exactly 10 models which achieved at least 86% accuracy, all of which are listed in Table 18.

Table 18. Top ten performing models on sample data

Model type	Learning rate	# of fully-connected layers	# of nodes per layer	Dropout rate	Validation accuracy
Classifier	0.005	4	50	0.1	87.86%
Regressor	0.005	3	50	0	87.27%
Classifier	0.005	4	10	0.2	86.70%
Classifier	0.01	4	40	0.2	86.61%
Regressor	0.005	2	30	0.1	86.59%
Regressor	0.005	4	50	0.1	86.49%
Regressor	0.005	3	50	0.1	86.29%
Classifier	0.005	2	30	0.2	86.07%
Classifier	0.01	3	30	0.2	86.07%
Regressor	0.005	2	40	0.1	86.06%

The best-performing SVM model reaches a validation accuracy of 83.90% (in the case where C=20 and gamma=50), so unfortunately they don't stack up against the Keras models. There may still be specific cases where SVMs are preferable however, so each backend prediction server would be better served to have at least one such model available to its model selector.

5 Discussion and Conclusions

5.1 Evaluation of Final Design

The objective of this project was to design a predictive system for the *Pathways* platform that can learn patterns between review ratings, candidates, and jobs/courses. The final design has three primary subsystems that satisfy all the functional and non-functional requirements of the project.

The Data Layer is built off the existing *Pathways* Flask API server and extends the existing MySQL database without losing any data, and a single git pull can upgrade an existing *Pathways* server to its *Pathways-Predictions* form in mere seconds, which satisfies FS1. It allows candidates to submit HTTP requests for suggestions of relevant jobs or courses on the platform, thus fulfilling FS2.

The Transport Layer encodes the data being sent between servers and maintains a very high throughput, able to not only handle more than 10,000 postings on the platform while making suggestions, but also doing so within a second of the user submitting the request, fulfilling NFS2 and NFS4. It includes a load balancer that distributes the volume of requests across multiple servers, and is highly fault-tolerant, able to handle the sudden failure of every single backend server except for one, fulfilling NFS5 and NFS6.

The Predictions Layer both trains on new review data and gives suggestions for recommended jobs or courses to the specific user requesting it. The input to the machine learning models is a combination of a specific user's information and valid job/course posts, so the output will reflect this and the suggestions will be both valid and personalized, fulfilling FS3 and FS4. New reviews are input to the predictive model as soon as they become available and don't interrupt the operation of the platform, fulfilling FS5. Several models are trained simultaneously and the best-performing one is used for any given request, resulting in a consistent model accuracy of over 75%, fulfilling FS6 and NFS1. These models also train very quickly, between 1-8 seconds, fulfilling NFS3.

Overall, the final design effectively meets all specifications and goals set by the team that can be determined at this time.

5.2 Use of Advanced Knowledge

This project contains many different components that require knowledge from various third and fourth year ECE courses. The necessary background in machine learning and neural network architecture, including data preparation and performance measures, are covered in ECE 457B. Concepts from ECE 356 are required in design of a relational database schema. Communication between machines over websockets and the use of transport layer protocols depended on knowledge from ECE 358. The process of distributing load across the web server and the detached ML server, ensuring uptime and resiliency, requires knowledge from ECE 454. Ensuring low latency and high scalability employed analysis of application bottlenecks and profiler-guided optimization from ECE 459.

5.3 Creativity, Novelty, Elegance

This work builds on existing frameworks for machine learning and web servers, but this design is creative and novel in the way that it will continuously train models with new labeled data and then dynamically select the best model at the time of a request. It also maintains a distributed service that can scale to large input sizes and maintain minimal latency with efficient database schemas while maintaining data consistency of its local cache. More technical novelty comes from how the data layer, predictions layer, and ML models all come together to add unique and meaningful improvements to a platform for online learning and job boards which broadens its appeal and ease of use.

The elegance of the project comes from the seamless integration of these modules with each other and the original system while maintaining a loose coupling. Each logical transformation of data is contained within its own individual layer, and the implementation details of any module is not known to any others. The application as a result is highly maintainable and scalable, and can be repurposed to almost any kind of supervised learning task based on user ratings. Finally, the abstraction of the predictive models into forms that can be easily compared or hooked into new systems is both creative and elegant.

5.4 Student Hours

Ben Chapman-Kish: 68 hours (2020-2021).
Ben Chapman-Kish: 186 hours (2022).

5.5 Potential Safety Hazards

As this is a software project, the risk to human health and property is minimal. Safety hazards would come in the form of security flaws that leak personal information, or allow an attacker to impersonate a user. Since the application uses OAuth login, the risk is very low, because it is virtually impossible for an attacker to compromise Google, Amazon, or other large enterprises. The website will be end-to-end encrypted with HTTPS, thus preventing any man-in-the-middle attacks.

Glossary

Internet and networking

TCP/IP: transmission control protocol/internet protocol
HTTP: hypertext transfer protocol, operates on TCP ports 80 or 443
Socket: more general TCP/IP protocol that can work on any port
Latency (or Ping): the time it takes for a single packet of information to be received by communicating machines
Throughput: the rate at which data can be processed by a service or network
API: application programming interface

Data formats and storage

Serialization: the process of encoding data into a form that can be sent on communication channels
MySQL: a database software that associates keys with unique values
JSON: JavaScript Object Notation, a human-readable data format
XML: eXtensible Markup Language, a human-readable data format
BCNF: Boyce-Codd Normal Form, the strongest normal form of database normalization which corresponds to absolutely no functional dependencies

Machine learning

ML: machine learning
CUDA: Compute Unified Device Architecture, a proprietary parallel computing platform by Nvidia widely used for GPU-enabled machine learning
ANN: Artificial Neural Network
SVM: Support Vector Machine
MNIST: Modified NIST database, a common machine learning dataset of handwritten digits
CIFAR-10: Canadian Institute for Advanced Research dataset, a common machine learning dataset of tiny images categorized among ten classes
RBF: radial basis function
ReLU: rectified linear unit
Feature: a parameter of data used as input for a neural network
Label: the expected output of a neural network for a given input
Classification: the task of assigning output classes for sets of input data
Regression: the task of estimating a function that maps inputs to outputs
Accuracy: the percentage of classification predictions that match the correct label
K-fold cross-validation: a performance-enhancing technique involving taking turns using portions of a dataset for training while preventing overfitting by using the remainder for validation

References

[1] S. Wickramasinghe, "Easiest and Hardest Developer Skills to Hire For in 2021," *crowdbotics.com*, Oct. 21, 2020. Accessed: June 14, 2022. [Online]. Available: https://www.crowdbotics.com/blog/ easiest-and-hardest-developer-skills-to-hire-for-in-2021

[2] S. G. Westlund and J. C. Hannon, "Retaining Talent: Assessing Job Satisfaction Facets Most Significantly Related to Software Developer Turnover Intentions," *Journal of Information Technology Management*, vol. 19, no. 4, 2008. Accessed: July 9, 2022. doi: 10.1.1.552.3878. [Online]. Available:
https://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.552.3878&rep=rep1&type=pdf

[3] S. Amar, "Python Serialization Benchmarks," *medium.com*, Dec. 31, 2018. Accessed: April 26, 2022. [Online]. Available: https://medium.com/@shmulikamar/ python-serialization-benchmarks-8e5bb700530b

[4] Y. Wu, L. Liu, C. Pu, W. Cao, S. Sahin, W. Wei, and Q. Zhang, "A Comparative Measurement Study of
Deep Learning as a Service Framework," *IEEE Transactions on Services Computing*, vol. 15, no. 1, July 18, 2019. Accessed: May 28, 2022. doi:10.1109/TSC.2019.2928551. [Online]. Available: https://arxiv.org/pdf/1810.12210.pdf

[5] C. M. L. Burnett, "Artificial neural network," *commons.wikimedia.org*, Dec. 27, 2006. Accessed: July 19, 2022. [Online Image]. Available: https://commons.wikimedia.org/wiki/ File:Artificial_neural_network.svg

[6] Z. Hasan, "Bias-variance trade-off," *zahidhasan.github.io*, Oct. 13, 2020. Accessed: July 19, 2022. [Online Image]. Available: https://zahidhasan.github.io/2020/10/13/ bias-variance-trade-off-and-learning-curve.html