RIGATonI: An R software for Rapid Identification of Genomic Alterations in Tumors affecting lymphocyte Infiltration

Running title: Identification of genomic variants and altered immune phenotypes in cancer

Raven Vella[1,2,3,4], Emily L. Hoskins[1,2,4], Lianbo Yu[4,5], Julie W. Reeser[1,4], Michele R. Wing[1,4], Eric Samorodnitsky[1,4], Leah Stein[1,2,4], Elizabeth G. Bruening[6], Anoosha Paruchuri[1,4], Michelle Churchman[7], Nancy Single[4], Wei Chen[4,8], Aharon G. Freud[4,8], Sameek Roychowdhury[1,4]

[1] Department of Internal Medicine, Division of Medical Oncology, The Ohio State University, Columbus, OH 43210

[2] Biological Sciences Graduate Program, The Ohio State University, Columbus, OH 43210

[3] Medical Scientist Training Program, The Ohio State University, Columbus, OH 43210

[4] Comprehensive Cancer Center, The Ohio State University, Columbus, OH 43210

[5] Department of Biomedical Informatics, The Ohio State University, Columbus, OH 43210

[6]The Bioinformatics Program, Loyola University Chicago, Chicago, IL 60660

[7] Aster Insights, Tampa, FL 33612

[8] Department of Pathology, The Ohio State University, Columbus, OH 43210

Corresponding author: Sameek Roychowdhury, Associate Professor, Department of Internal Medicine, Division of Medical Oncology, Comprehensive Cancer Center and The James Cancer Hospital, The Ohio State University, Columbus, OH 43210, USA. Tel: +1 614-685-5842; email: Sameek.roychowdhury@osumc.edu

Key words: Genomics, immune infiltration, machine learning, tumor immunity, bioinformatics

Word count: 4536 words

ABSTRACT

Tumor genomic alterations have been associated with altered tumor immune microenvironments and therapeutic outcomes. These studies raise a critical question: are there additional genomic variations altering the immune microenvironment in tumors that can provide insight into mechanisms of immune evasion? This question is the backbone of precision immuno-oncology. Current computational approaches to estimate immunity in bulk RNA sequencing (RNAseq) from tumors include gene set enrichment analysis and cellular deconvolution, but these techniques do not consider the spatial organization of lymphocytes or connect immune phenotypes with gene activity. Our new software package, Rapid Identification of Genomic Alterations in Tumors affecting lymphocyte Infiltration (RIGATonI), addresses these two gaps in separate modules: the Immunity Module and the Function Module. Using pathologist-reviewed histology slides and paired bulk RNAseq expression data, we trained a machine learning algorithm to detect high, medium, and low levels of immune infiltration (Immunity Module). We validated this technique using a subset of pathologist-reviewed slides not included in the training data, multiplex immunohistochemistry, flow cytometry, and digital staining of The Cancer Genome Atlas (TCGA). In addition to immune infiltrate classification, RIGATonI leverages another novel machine learning algorithm for the prediction of gain- and loss-of-function genomic alterations (Function Module). We validated this approach using clinically relevant and function-impacting genomic alterations from the OncoKB database. Combining these two modules, we analyzed all genomic alterations present in solid tumors in TCGA for their resulting protein function and immune phenotype. We visualized these results on a publicly available website. To illustrate RIGATonI's potential to identify novel genomic variants with associated altered immune phenotypes, we describe increased anti-tumor immunity in renal cell carcinoma tumors harboring 14q deletions and confirmed these results with previously published single-cell RNA sequencing. Thus, we present our R package and online database, RIGATonI: an innovative software for precision immuno-oncology research.

Identification of genomic variants and altered immune phenotypes in cancer

Immune surveillance is crucial for the eradication of cancer cells[1]. Investigating factors impacting the tumor immune microenvironment can aid in understanding this process[1]. The tumor microenvironment is composed of tumor cells, along with surrounding immune cells, stroma cells and tissue matrix, and microbiota[1]. Each of these elements vary across patients and tumor types, producing a spectrum of immune phenotypes and, consequently, clinical outcomes to immunotherapy[1]. In particular, the field of precision immuno-oncology requires approaches designed to discover relationships between tumor genomic alterations and their associated microenvironments[2, 3]. There are several techniques to assess the tumor immune microenvironment using bulk tumor RNAseq data; however, there are no approaches designed to rapidly detect associations between tumor genomic alterations and the quality of tumor inflammation in big data repositories. Thus, there is a need to develop computational tools specifically designed to assess immunity in large databases of tumor transcriptomes.

Current methods for detecting altered immunity in tumors (**Figure 1A**) include gene set enrichment analyses (e.g., ImSig[4], Thorsson *et. al.*[5]), cellular deconvolution techniques (e.g., MCP-counter[6], quantiseqR[7], CIBERSORT[8]), and ensemble results provided in databases (e.g., TCIA[9], TIMER2.0[10], TIMEDB[11]). Gene set enrichment analysis involves assessing the expression of immune-related gene sets and then clustering cancers based on the results. This is routinely employed in large studies of immunotherapy outcomes to reveal the tumor features that underscore interpatient differences[12, 13]. These approaches are often not optimized for application to data outside the study within which they are built because they are not intended for robust analysis across databases. Alternatively, cellular deconvolution tools attempt to emulate flow cytometry or immunohistochemistry by estimating the number of specific immune cells in a sample[6-8]. Importantly, cellular deconvolution does not make clear distinctions between immune phenotypes broadly and rather leaves interpretation up to the user. Databases of cellular deconvolution results have emerged, which include ensemble analyses across The Cancer Genome Atlas (TCGA) and other sources[10, 11]. These databases provide information about associations between genomic alterations and immune phenotypes; however, they do not separate analyses based on the functional status of the genomic alteration instead combining all alterations in a gene of interest regardless of their molecular impact[9-11]. None of these techniques were specifically developed to classify immune phenotypes and discern associations with functionally relevant tumor genomic alterations.

To address these gaps, we developed Rapid Identification of Genomic Alterations in Tumors affecting lymphocyte Infiltration (RIGATonI). RIGATonI is composed of two machine learning modules to connect functional genomic alterations (Function Module) and altered immune phenotypes (Immunity Module) in an unbiased manner. Our approach is distinct from currently available bulk RNAseq methods in four ways: 1) model training with pathologist-defined immune infiltration as gold standard, 2) consideration for immune cell spatial characteristics (e.g., tertiary lymphoid structures, dispersion characteristics, and degree of infiltration), 3) rapid, robust, and precise immune phenotype classification across big data resources, and 4) gene candidate filtering that evaluates protein function prediction rather than gene mutation status alone (**Figure 1A**). To build and validate RIGATonI, we combined histologic features identified by computational staining[14], pathologist-defined immune infiltration, protein-protein interaction networks[15], genomic data, proteomic data, and transcriptomic data (**Figure 1B**).

First, we built the Immunity Module to predict immune phenotypes using bulk RNAseq expression. Contrasting other approaches, this module was not built using exclusively immune-related genes; instead, we performed unbiased feature selection to determine the best predictors (n=114) of tumor immunity (**Figure 1A**). We validated and fine-tuned our approach using manually reviewed tumor histology by pathologists, computational staining[14], immunohistochemistry[16], and flow cytometry[16] (**Figure 1A**). Next, we developed the Function Module which can accurately predict the function of genomic alterations (copy number alterations, single nucleotide variations, and structural variations) from bulk RNAseq expression. We validated this module using data from the largest collection of functionally impactful, clinically relevant genomic alterations in cancer: OncoKB[17]. These two modules were combined to uncover connections between all the genomic alterations in solid tumors in TCGA and the immune phenotypes of samples harboring these alterations (**Figure 1B**). We created an interactive visualization interface (https://rigatoni.osc.edu) to help researchers access our TCGA analysis results for individual genes (**Figure 1B**).

A

	Gene Set Enrichment	Cellular Deconvolution	RIGATonI Immunity Module
Goal	Find subgroups of cancer	Predict number of each immune cell subtype	Determine degree of inflammation in a tumor sample
Feature Selection	Known immunity gene sets	Known immunity gene sets	Agnostic gene selection with machine learning
Protein Validation		Correlation with flow cytometry data or liquid cell suspensions	Statistical comparison of flow cytometry and multiplex immunohistochemistry
Clinical Validation			Pathologist annotation of lymphocyte infiltration

B

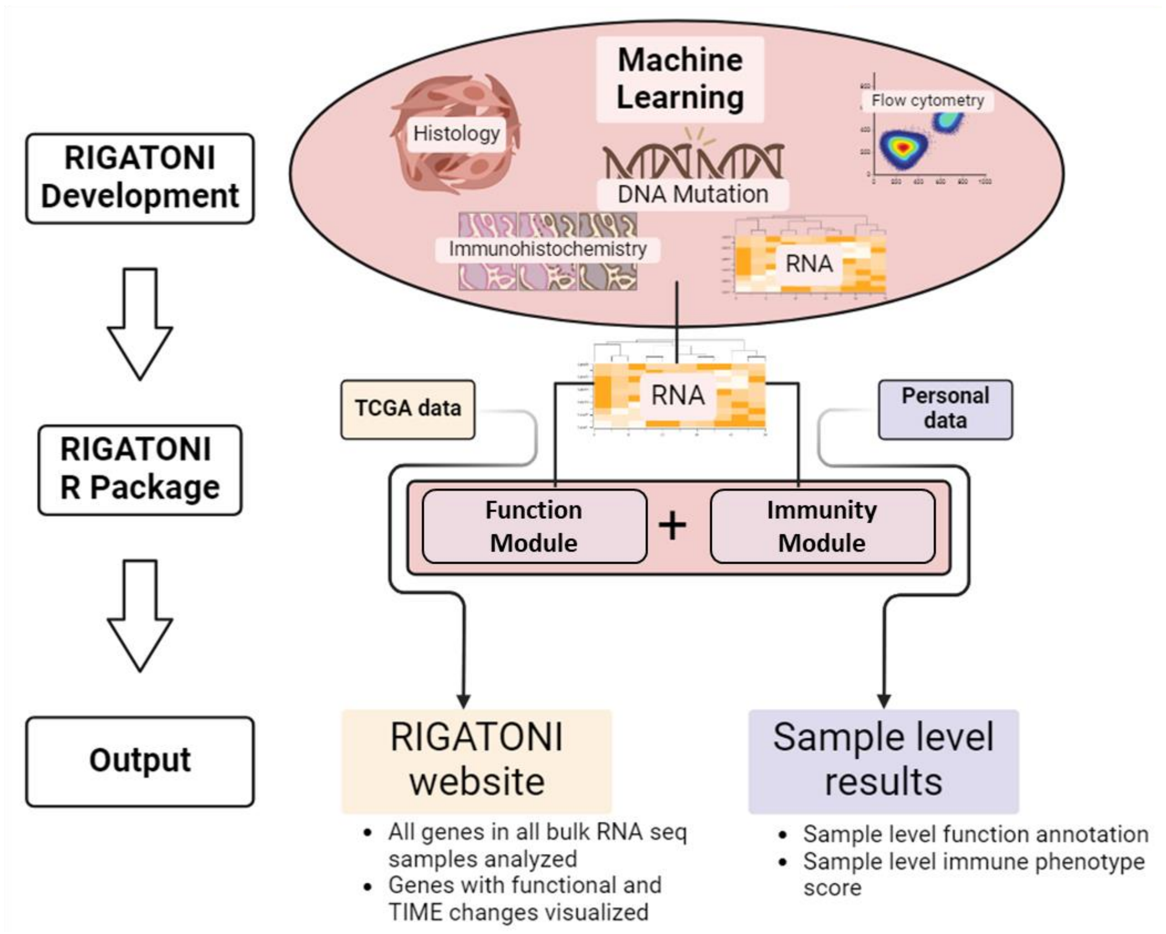

Figure 1. Overview of RIGATonI as a novel agnostic approach to measure immune infiltration from bulk tumor RNASeq. A. Current approaches to estimate immunity from bulk tumor RNAseq include gene set enrichment[4, 5] and cellular deconvolution[6-8] based methods. In contrast, RIGATonI utilizes a gene agnostic approach to classify immune cell infiltration in tumors by training on pathologist-annotated digital slides. **B.** RIGATonI enables identification of candidate genomic alterations associated with altered immune infilration in subsets of cancer through evaluation of gene expression, genomic alterations, pathologist-classified tumors, and protein validation. The Immunity Module assesses immune infiltration from bulk RNAseq expression for individual samples. The Function Module predicts the function (loss or gain) of individual samples for a given gene of interest using bulk RNAseq expression.

Additionally, we built an R package which can be used to perform the RIGATonI analyses on any samples of interest (**Figure 1B**).

To demonstrate the applications of the RIGATonI software, we explored a novel connection between 14q deletion in renal cell carcinoma and increased anti-cancer immunity discovered by RIGATonI's analysis of TCGA. Using bulk RNA sequencing and single cell RNA sequencing from TCGA and Yu *et. al.*[18] respectively, we show increased infiltration of CD8+ T cells, decreased pro-tumor immune checkpoint signatures, and increased CD8+ T cell proliferation, cytotoxicity, and inflammation.

Together, these results introduce RIGATonI: a unique and powerful tool designed with machine learning to identify immunologically impactful genomic alterations in cancer.

RESULTS

RIGATonI predicts immune phenotypes by utilizing histology and bulk RNAseq with high accuracy

RIGATonI's Immunity Module was trained and validated using a comprehensive, pan-cancer dataset (OSU-ORIEN dataset) from The Ohio State University (OSU) including digital histology paired with bulk RNAseq (sequenced by Oncology Research Information Exchange Network, ORIEN). To ensure the OSU-ORIEN dataset included sufficient low, medium, and high immune phenotypes, we used a preliminary version of our machine learning algorithm developed using computational staining[14] output from TCGA. We succeeded in doing so and produced a training data set of 403 tumors across 22 different cancer types (**Supplemental Figure 1**). Digital histology slides from these tumors were reviewed independently by two pathologists and were classified into low, medium, or high immune infiltration groups (**Figure 2A**). Pathologists used a semi-quantitative approach to estimate the percentage of tumor area occupied by lymphocytes. They also considered the distribution of these lymphocytes throughout the tumor area (e.g., deeply penetrating, semi-penetrating, or peripheral), and the overall quality of inflammation (e.g., presence or absence of tertiary lymphoid structures, signs of cytotoxic killing of tumor cells). To build the final model, we evaluated six different models using two different machine learning approaches and pathologist annotations both together and separately (**see Methods**). We selected genes for immune phenotype prediction within the Immunity Module using ElasticNet[19], which yielded 114 transcriptomic features (**Supplemental Table 1**). All bulk RNAseq expression for these 114 genes were extracted and subsequent analyses utilized only this data. An XgBoost[20] algorithm was trained using 334 of 403 tumors with Bayesian parameter optimization[21] to classify tumors' immune infiltration. We assessed the accuracy of the algorithm in a variety of ways.

First, we evaluated the accuracy of RIGATonI to classify 69 blind-set samples (**Figure 2B**). The overall accuracy of the model was 71.01% (95% confidence interval 58.84%-81.31%). The balanced accuracy was higher for tumors with high and low infiltration (82.04% and 82.58%, respectively) compared to those with medium infiltration (63.4%). A similar trend was observed for sensitivity and specificity (**Supplemental Table 2-4**). The accuracy was significantly different from the no information rate ($p<0.01$), indicating that the overall accuracy is significantly better than what could be achieved from random chance. We also failed to reject the null hypothesis of McNemar's test, which indicates there is insufficient evidence that the predictions made by the algorithm are different from the true phenotypes ($p>0.05$). Detailed statistics and a confusion matrix of the results are available in **Supplemental Table 2-4**.

RIGATonI's Immunity Module corresponds with mIHC and flow cytometry features of increased lymphocyte infiltration

We validated the Immunity Module with a set of 32 gastric tumors from a recent study which provided matched multiplex immunohistochemistry (mIHC), flow cytometry, and bulk RNAseq[16]. Tumors classified "high" by RIGATonI (RIGATonI-high) had higher immune cell counts detected by mIHC (**Figure 2C**). Further analysis revealed that this increase was mainly due to a greater number of CD3+ cells in RIGATonI-high tumors compared to RIGATonI-low tumors (**Figure 2D/Supplemental Figure 2A)**. Further, flow cytometry data from the same study confirmed our findings. RIGATonI-high tumors showed a significantly higher percentage of lymphocytes (specifically CD3+ cells) compared to RIGATonI-low tumors (**Figure 2E/Supplemental Figure 2B)**. Additionally, we observed a significant increase in CD8+ T cells measured by mIHC in RIGATonI-high tumors compared to RIGATonI-low tumors (**Figure 2F**).

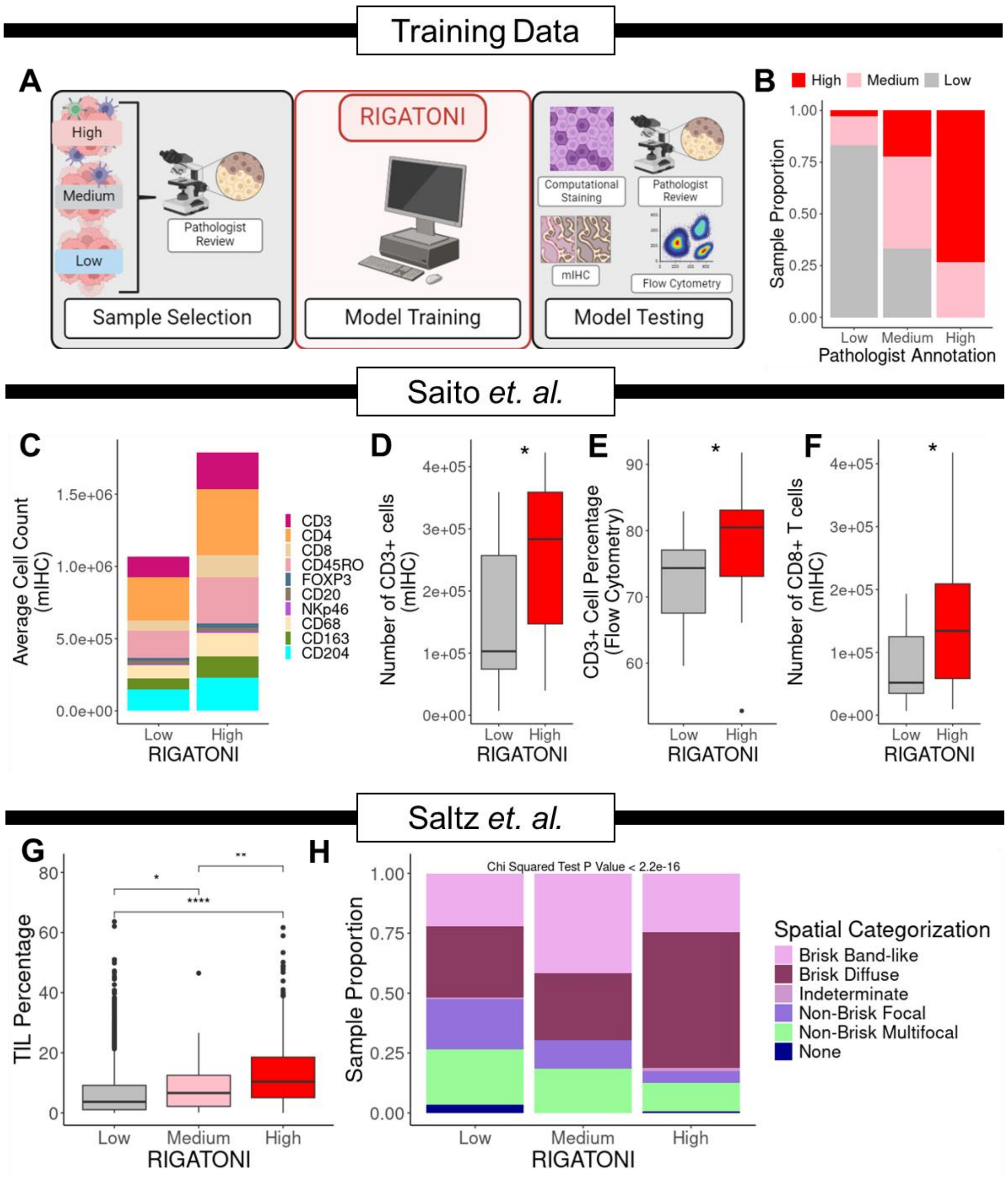

Figure 2. Development and validation of RIGATONI immune infiltrate classification. A. To train RIGATonI, we selected tumor samples based on a preliminary algorithm developed to identify a range of low to high lymphocyte infiltration phenotypes. Tumor samples were selected across diverse cancer types and two pathologists independently classified them as high, medium, or low degrees of infiltration. We built a machine learning model to predict these annotations and validated with a variety of data types. **B.** The accuracy of the model (Y-axis) was validated using a blind dataset (X-axis) not used for training data. RIGATonI-high annotations were 82% accurate for pathologist high and low annotations. RIGATonI-medium predictions were 63% accurate. **C.** Next, we investigated an independent dataset of gastric tumors with mIHC, flow cytometry, and tumor RNAseq data[16]. RIGATonI -high and -low samples corresponded to immune cell subsets as measured by mIHC. **D.** The counts of CD3+ T cells detected by mIHC were significantly increased in RIGATonI-high samples. **E**. Flow cytometry of these tumors demonstrated an increase in CD3-positive lymphocytes in RIGATonI -high vs. -low samples. **F.** The counts of CD8+ T cells detected by mIHC were significantly increased in RIGATonI-high samples. **G.** We also investigated our algorithm's association with histologic features detected by convolutional neural networks in 5,202 tumors from TCGA[14]. The percentage of tumor infiltrating lymphocytes were measured by Saltz *et. al.*[14] and corresponded with RIGATonI -low, -medium, and -high classifications. **H**. We evaluated spatial characteristics of lymphocyte infiltrates using Saltz *et. al.*[14] approach. The overall distribution of spatial characteristics is significantly different across RIGATonI -low, -medium and -high subsets. RIGATonI-high samples often displayed brisk diffuse lymphocyte patterns shown in maroon**.** Significance values: p≤0.05: *, p≤0.01: **, p≤0.001: ***, p≤0.0001: ****.

RIGATonI phenotypes aligned with findings from computational staining of lymphocytes

We applied the RIGATonI Immunity Module to 5,202 tumors from TCGA and examined the output of an orthogonal approach for computational staining of digital pathology slides[14]. Saltz *et. al.*[14] developed a convolutional neural network to detect the percentage and distribution of lymphocytes on digital histology images. First, we compared RIGATonI classifications to the predicted lymphocyte percentage on the slide (**Figure 2G**). We saw that RIGATonI-high tumors displayed significantly greater lymphocyte percentages compared to RIGATonI -medium or -low tumors (**Figure 2G**). Similarly, RIGATonI-medium tumors displayed significantly greater lymphocyte percentages compared to RIGATonI-low tumors (**Figure 2G**). Next, we compared RIGATonI classifications to five patterns of spatial attributes of tumors described by Saltz *et. al.*[14]: brisk band-like, brisk diffuse, indeterminant, non-brisk focal, non-brisk multifocal, or none (**Figure 2H**). We observed a significant difference in the spatial arrangements of lymphocytes between RIGATonI classifications (**Figure 2H**). Lymphocytes from RIGATonI-high tumors were more likely to have a brisk diffuse arrangement than the population ($p<0.01$), defined by a broad distribution of many lymphocytes throughout the histology slide[14]. RIGATonI-medium tumors display brisk band-like arrangements more often than the population ($p<0.01$). Brisk band-like infiltration patterns indicate the lymphocytes are clustered in a band across the slide, but that there are many lymphocytes[14]. RIGATonI-low tumors displayed a higher proportion of non-brisk focal lymphocyte arrangements ($p<0.01$). Non-brisk focal arrangements indicated negligible numbers of lymphocytes in a handful of locations across the image[14]. In summary, RIGATonI-high tumors exhibited extensive lymphocyte infiltration throughout the tissue slide, while RIGATonI-low tumors showed limited infiltration in isolated and/or scattered spots **(Figure 2H)**.

RIGATonI includes an innovative Function Module which can accurately classify genomic alterations with molecular effects using protein-protein interaction networks

The Function Module first uses the STRING[15] protein-protein interaction database to identify proteins which have direct and validated interactions with the protein of interest (**Figure 3A**). Both proteins which act on the protein of interest (upstream proteins) and proteins on which the protein of interest acts (downstream proteins) are considered. Next, we collect the gene names corresponding to the proteins into two lists. These two gene sets serve different purposes: upstream genes allow assessment of alterations which impact the expression level of the gene of interest; downstream genes allow us to assess the impact of alterations which may change the activity level of the gene of interest. We built a generalized linear model over a Poisson distribution using the extracted RNAseq counts of upstream and downstream genes using samples with no alteration in the gene of interest. These two models are then saved and used to predict the RNAseq counts of the gene of interest within the mutant samples. 95% prediction intervals are created for both the upstream and downstream models. Mutant samples with true counts below the 95% prediction interval of either model are classified as “LOF”. Conversely, samples with counts exceeding the 95% prediction interval are classified as “GOF”. Samples within the 95% prediction interval of both models are annotated as “unknown” (**Figure 3A**).

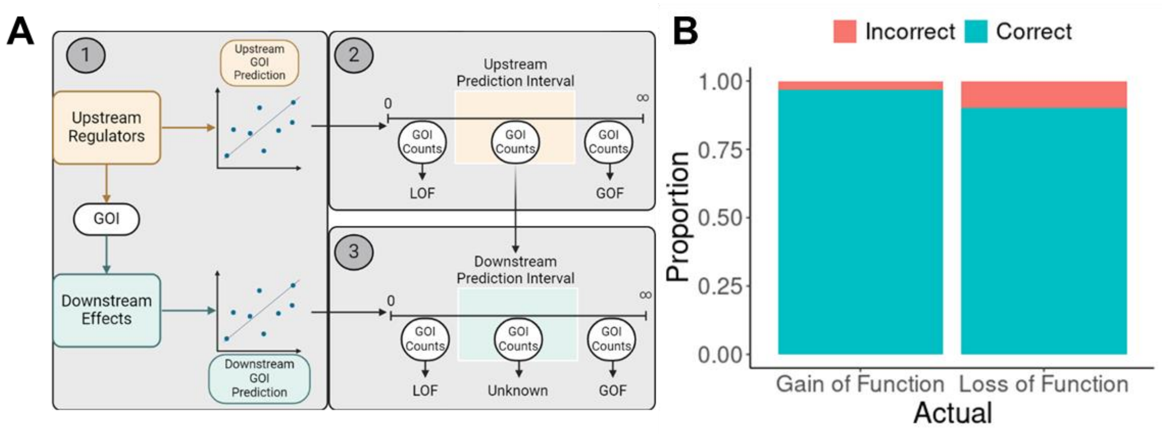

Figure 3: Function Module development and testing. A. The Function Module was developed using upstream regulators and downstream targets of a given gene of interest from the STRING[15] database. Parallel linear models over a Poisson distribution were built. First, the model of the upstream regulators was assessed. Next, if the expression of the gene of interest falls within the prediction interval, gene expression predicted by the downstream targets was assessed. **B.** The Function Module was assessed using OncoKB[17] as a ground truth. There were 291 genomic alterations with corresponding annotations in OncoKB[17] and 96.8% of the gain-of-function calls (60/62) and 90.4% of the loss-of-function calls (207/229) were correctly classified.

To assess the Function Module’s accuracy, we analyzed genomic alterations from a total of 1008 oncogenes and tumor suppressor genes annotated by OncoKB[17]. When applied to 10,464 tumors from the TCGA, our algorithm successfully identified 400 GOF alterations and 966 LOF genomic alterations (SNVs, structural variations, and copy number variations) within these 1008 genes. Genomic alterations harbored by fewer than five tumors were not assessed. Fusions were excluded due to the high potential for false positives as many tumors harbored multiple fusions and imprecise breakpoints.

Of the 400 candidate GOF alterations uncovered, 62 were annotated in the OncoKB[17] database. The algorithm correctly categorized 60 of 62 as GOF, and two were incorrectly categorized as LOF, yielding an accuracy of 96% for GOF (**Figure 3B**). Notably, 84.5% of the alterations annotated by the algorithm had no information available in OncoKB[17]; most of these variants were whole gene amplifications (**Supplemental Table 5).** Similarly, we validated the accuracy of this algorithm for LOF alterations. Of the 966 LOF alterations uncovered, only 229 were annotated in the OncoKB[17] database. The algorithm correctly categorized 207 as LOF, while 22 were incorrectly categorized as GOF, yielding an accuracy of 90% (**Figure 3B**). Further, 76% of the candidate LOF alterations annotated by the algorithm had no information available in OncoKB[17]; most of these variants were whole gene deletions or premature truncations (**Supplemental Table 5)**. Overall, these findings demonstrate our algorithm's effectiveness in accurately classifying known GOF and LOF mutations while also identifying novel variants that are not well described in existing databases.

RIGATonI's TCGA analysis is available for exploration online

Using the RIGATonI modules outlined above, we compiled and analyzed all genomic alterations (structural variations, gene fusions, point mutations, and copy number alterations) in TCGA. In total, RIGATonI identified 7,410 genomic alterations with possible immune effects among 5,746 genes. To determine the number of novel results among the RIGATonI output, we performed text mining on 226,093 abstracts mentioning "cancer" and "immunity" published between June 22nd, 2010, and June 22nd, 2023. We discovered that 2,773 (48%) of the RIGATonI output genes had not been previously connected to cancer immunity (**Figure 4A**). Only 72 genes (1%) had been mentioned in cancer immunity abstracts more than 100 times (**Figure 4A**). All results are available online at https://rigatoni.osc.edu/. Users select a gene of interest to explore and can subset output with alterations or cancer types of interest on the home page (**Supplemental Figure 3**). In the Transcriptomics page, the user can explore the expression levels of different genes across patient groups (**Supplemental Figure 4**). Finally, we provide cellular deconvolution results from quantiseqR[7] on the Immunity page (**Supplemental Figure 5**). This

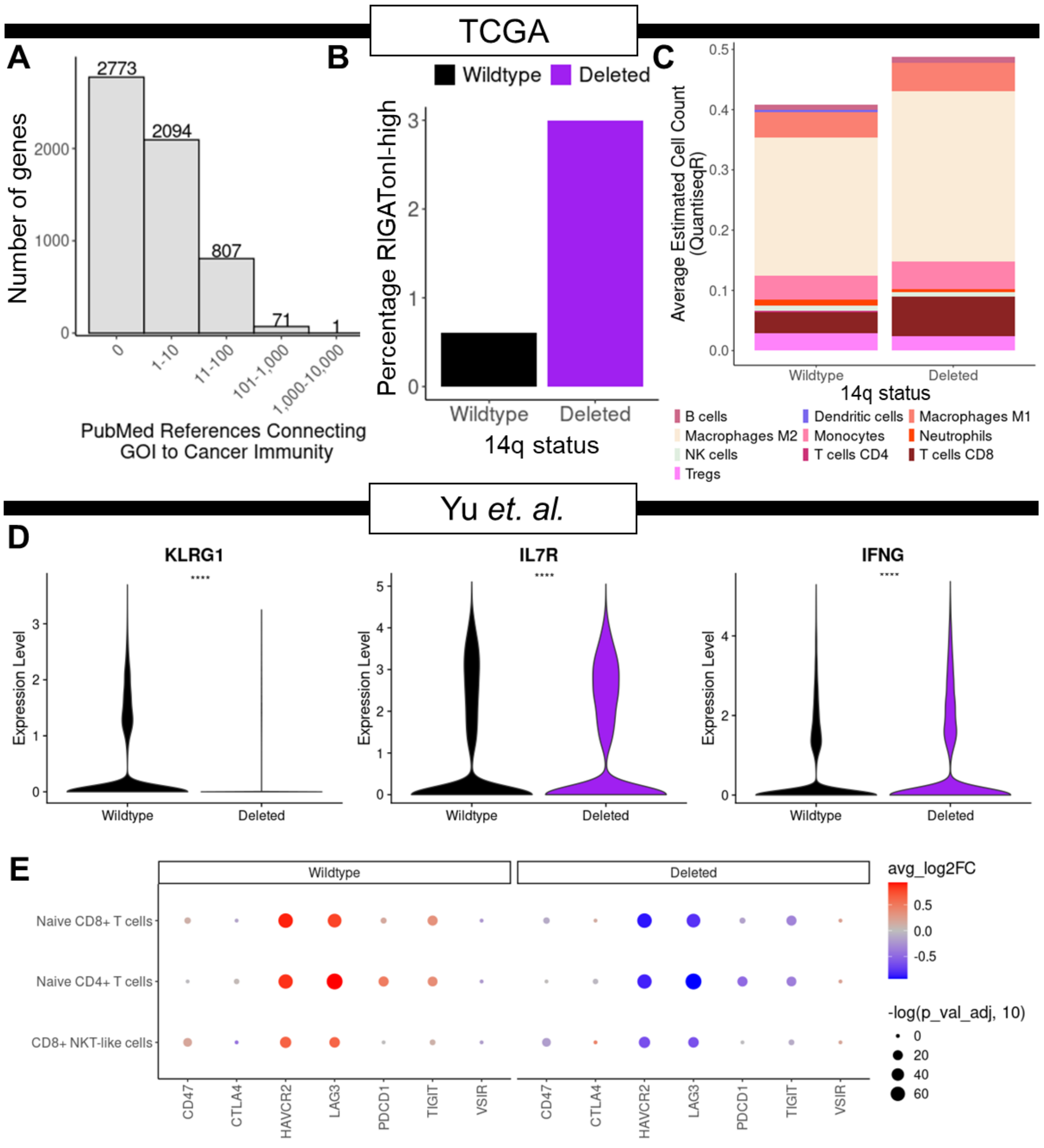

Figure 4. RIGATonI identifies novel genomic alterations of interest including 14q deletion associated with an increased immune infiltrate and effector CD8+ T cells in renal cell carcinoma. A. Through analysis of TCGA and text mining of PubMed, 48% (n = 2773) of genes harboring RIGATonI-identified genomic alterations (n = 5746) have never been associated with tumor immunity. **B.** As an example, RIGATonI identified 14q deletion in renal cell carcinoma (RCC) samples which corresponded to an increased immune infiltration compared to wildtype tumors. **C.** Using quanitseqR[7], we corroborate our finding that there is a broad increase in immune cells in 14q-deleted RCC tumors**. D.** With 19 scRNAseq RCC experiments, we investigated markers of CD8+ T cell exhaustion and a "cold" immune microenvironment. CD8+ T cells from 14q-deleted tumors displayed decreased exhaustion marker *KLRG1* and increased anti-tumor immunity markers *IL7R* and *IFNG*[22-24] **E.** Immune checkpoints, which are thought to promote tumor growth, are downregulated across all T cells studied in 14q-deleted tumors compared to wildtype tumors. These checkpoint receptors include *CD47*, *CTLA4*, *HAVCR2*, *LAG3*, *PDCD1*, *TIGIT*, and *VSIR* [25, 26]. Significance values: p≤0.05: *, p≤0.01: **, p≤0.001: ***, p≤0.0001: ****

website is intended to assist researchers in understanding how the function of their gene of interest impacts immunity in an unbiased and rapid manner.

RIGATonI identifies genomic alteration 14q deletion's association with altered immune infiltration in renal cell carcinoma

Exploring novel results from the TCGA analysis, we uncovered a group of genes (n=207) all located on chromosome 14q which consistently showed the same patterns of deletion, in the same patients, and were RIGATonI-high. Further investigation led us to discover that these alterations were part of a larger chromosomal arm deletion of 14q. We observed this pattern in many cancer types, but one of the strongest relationships was in renal cell carcinoma (RCC). 3% of RCC patients harboring a 14q deletion were highly infiltrated compared to 0.6% of 14q wildtype RCC patients (**Figure 4B**). We also corroborated these results using quantiseqR[7] where 14q-deleted tumors displayed an increased immune infiltration compared to wildtype (**Figure 4C**). Although these two bulk RNAseq methods show the same pattern, we also evaluated 14q deletion in an orthogonal dataset with 19 RCC tumors with single cell RNAseq[18]. Using copyKat[27], we discovered 8/19 patients harbored a 14q deletion. We evaluated the immune cell compartment of these samples and observed that CD8+ T cells from 14q-deleted tumors displayed decreased *KLRG1*, increased *IL7R*, and increased *IFNG* expression (**Figure 4D**). This pattern is indicative of increased T cell proliferation, cytotoxicity, and T cell-mediated anti-tumor immunity among CD8+ T cells from 14q-deleted tumors[22-24]. We also investigated the expression of immune checkpoint genes expressed on T cells (*HAVCR2*, *TIGIT*, *LAG3*, *PDCD1*, *VSIR*, and *CD47*) (**Figure 4E**)[25, 26]. Across all T cells studied, expression of pro-tumor checkpoint genes is decreased in 14q-deleted tumors (**Figure 4E**). The largest differences were in *LAG3* (LAG-3) and *HAVCR2* (TIM-3) (**Figure 4E**). CD4+ T cells were the only cell type to have a significant decrease in *PDCD1* (PD-1) within 14q-deleted tumors (**Figure 4E**). Together these results indicate that 14q-deleted tumors contain more activated CD8+ T cells than wildtype as well as fewer features of T cell exhaustion and pro-tumor immune activity.

DISCUSSION

Identification of genomic variants and altered immune phenotypes in cancer

RIGATonI is a novel tool to discover precision immuno-oncology targets using bulk RNAseq data in two distinct modules: the Immunity Module (**Figure 2A**) and the Function Module (**Figure 3A**). The Immunity Module addresses two key gaps not considered by current approaches 1) degree of immune cell dispersion alongside infiltration, and 2) immune phenotyping across big data resources without user interpretation. The Immunity Module uses a machine learning approach built with paired bulk RNAseq from pathologist reviewed histology slides (**Figure 2A)**. To our knowledge, this strategy has never been implemented before. The Function Module classifies genomic alterations as LOF or GOF based on bulk RNAseq features (**Figure 3A)**. RIGATonI is the first approach we are aware of for estimating immunity which also estimates gene-level function. RIGATonI was employed across the TCGA database and interesting results are visualized on our website (https://rigatoni.osc.edu/) for public use. Finally, we demonstrated that these two modules can be combined to discover novel connections between genomic alterations and immunity through further investigation of 14q deletion in renal cell carcinoma (RCC) scRNAseq datasets. In summary, RIGATonI is a unique machine learning approach specifically designed for precision immuno-oncology.

The Immunity Module considers features of cancer immunity not considered by available software tools. Currently, gene set enrichment, cellular deconvolution, and ensemble approaches for estimating immunity from bulk RNAseq have not been compared to benchmark pathologist review of tumor histology for degree and quality of lymphocyte infiltration (**Figure 1A**)[4-8]. To build RIGATonI's Immunity Module, we asked pathologists to review tumor histology slides for degree of infiltration considering not only the absolute number of lymphocytes, but also the presence or absence of tertiary lymphoid structures (TLS), signs of cytotoxicity, and the general distribution across the tumor slide (deeply penetrating the tumor or on the periphery). By incorporating these nuanced features into our algorithm training and validation, RIGATonI benchmarks aspects of immunity not evaluable with either gene set enrichment or cellular deconvolution tools[4-8]. Furthermore, both cellular deconvolution and gene set enrichment utilize genes with known connections to cancer immunity (**Figure 1A**). These genes are selected either through literature review or single cell atlases[5]. Unfortunately, this approach does not allow for evaluation of *de novo* mechanisms that impact tumor immunity. To address this gap, RIGATonI's Immunity Module uses expression of just 114 genes selected in an unbiased manner (**Supplemental Table 1**). These genes consistently predict immune phenotypes across cancer types and databases (**Figure 2B-H**). In the future, we will explore why this novel gene list can reproducibly predict cancer immune infiltration.

The Function Module's prediction of protein-level effects of genomic alterations from bulk RNAseq expression data is also novel. Most genomic alterations in tumors are variants of unknown significance[28]. Therefore, a functional measure of associated gene activity can help to prioritize alterations that are most likely to have a biological or immunological impact. eVIP2 is the only other method designed to determine the function of genomic variants using bulk RNAseq[29]. Importantly, eVIP2 was validated using only two alterations in the same gene of interest, whereas our validation set demonstrated remarkable accuracy involving 291 different alterations across 207 different genes (**Figure 3B**)[29]. VIPER, another R package, utilizes cell specific "regulons" for protein activity prediction and did assess scores for some genomic variants of interest; however, it was not designed to analyze the functional impact of genomic alterations directly[30]. RIGATonI is the only available bulk RNAseq method to assess protein function from gene expression validated with several hundred genomic alterations across many different genes. In contrast to DNA-based annotation of mutations, the Function Module has several benefits. First, although we applied the Function Module to groups with differing genomic alteration status, the module assesses the function of a gene of interest in a testing group compared to a control group. Thus, RIGATonI could be applied to any subsets of any other feature (e.g. methylation, alternative splicing, treatment, etc.) (**Figure 3A**). Second, when evaluating genomic alterations, RIGATonI can assess novel variants and mechanisms of altered expression. Third, not all patients with the same genomic alteration experience identical molecular effects. The Function Module predicts the overall effects of alterations and makes specific predictions for each sample, considering its unique molecular characteristics (**Figure 3A**). These unique features make RIGATonI's Function Module an effective tool for multi-omic research.

Like many computational approaches, RIGATonI is limited by sample size considerations. By using pathologist assessment of tumor histology rather than computer vision as a ground truth for the Immunity Module, we do limit the sample size of our training data. New approaches like Lunit SCOPE IO[31] and that of Saltz *et. al.*[14] use deep learning techniques to estimate immune cells on digital pathology slides. Lunit SCOPE IO was trained on >17,000 H&E images across 24 cancer types[31]. This scale of training data would be impractical using manual pathologist review. However, it is widely agreed that physician benchmarking is the gold standard against which machine learning approaches should be measured[32]. Therefore, we believe the quality of the training data classifications compensated for comparatively smaller sample size. Another sample size concern arose when validating the Function Module. Validation of RIGATonI's Function Module only took place on alterations with ≥5 instances of a candidate alterations to ensure that statistical tests could be performed to determine the functional status. In the pan-TCGA analysis, we did not group together early terminations or different point mutations occurring at the same locus out of an abundance of caution (early terminations at different loci may have different effects; different point mutations at the same locus may have different effects). Without detailed knowledge of all 17,000 genes analyzed, we approached the first version of RIGATonI conservatively. Despite these sample size concerns, our robust validations give us confidence in our tool's performance.

RIGATonI identified 14q deletion as a potential novel biomarker for increased anti-tumor immunity in renal cell carcinoma (RCC). 14q deletion has previously been identified as a negative prognostic indicator in RCC; however, these studies were done prior to the broad adoption of PD-1/PD-L1 immunotherapy in RCC[33, 34]. RIGATonI indicates 14q deletions are associated with a highly infiltrated immune microenvironment in TCGA (**Figure 4B**). These results were further supported using the cellular deconvolution tool quantiseqR[7] which demonstrates enhanced immunity in 14q-deleted samples (**Figure 4C**). We were able to orthogonally assess 14q through analysis of scRNAseq data for 19 RCC patients[18]. We first explored the CD8+ T cells between 14q-deleted and wildtype tumors. CD8+ T cells from 14q-deleted tumors displayed evidence of superior cytotoxicity, increased release of interferon-gamma, and increased proliferation[22-24]. We also explored whether pro-tumor immune checkpoint receptors and ligands were more highly expressed in T cells from 14q-deleted vs wildtype tumors. Wholistically, we see that pro-tumor immune checkpoint receptors are decreased in 14q-deleted tumors compared to wildtype tumors across all cell types explored (**Figure 4E**)[25, 26]. Identification of 14q deletion in RCC demonstrates a successful application of RIGATonI to discover genomic alterations associated with altered tumor immunity.

In summary, RIGATonI is a powerful software leveraging tumor RNAseq in novel ways to accelerate discoveries in precision immuno-oncology research. RIGATonI can be applied across big data sources to understand tumor-driven mechanisms of immunity, identify novel biomarkers for immunotherapy treatment, and discover novel drug targets for future immunotherapies. In summary, we are pleased to introduce RIGATonI: an innovative approach for discovery novel genomic variants with associated altered immune phenotypes.

ONLINE METHODS

Tumor sequencing data. All data shown here were previously published or made available by a big-data source (described below). Secondary analyses were performed on the Pitzer cluster R studio v4.3.0 within Ohio Supercomputer Center (https://www.osc.edu/). Data sources include The Cancer Genome Atlas (TCGA) and the Oncology Research Information Exchange Network (ORIEN). RNAseq count data were downloaded from TCGA using GenomicDataCommons[35] and batch corrected with ComBat-seq[36] using institution of origin as the batch. Genomic variant calling data was downloaded in the form of combined .maf files. Copy number alteration data were downloaded in the form of gene based raw copy number. Finally, all whole genome .bam files were downloaded from TCGA and then processed with parliament2[37] using Delly[38], Manta[39], breakdancer[40], and breakseq[41] to assess for structural variants. Results from parliament2[37] were combined using SURVIVOR[42] with default settings. RNAseq data was also obtained from ORIEN and batch corrected with ComBat-seq[36] according to the RNAseq batch information made available through Aster Insights. Copy number alterations were downloaded from ORIEN in the form of gene based raw copy number. Demographic information was downloaded from ORIEN as well. ORIEN data is managed by Aster Insights, requests for this data should be sent to Aster Insights.

RIGATONI immune phenotyping algorithm development and validation with pathologist review. Two pathologists independently reviewed 403 tumor slides assessing lymphocyte infiltration characteristics. Pertinent characteristics included percentage of space not occupied by tumors or stroma, which was occupied by lymphocytes, dispersion of lymphocytes within tumor, and presence or absence of tertiary lymphoid structures. Taking into consideration all these characteristics, each pathologist annotated the slide either high, medium, or low infiltration. Paired RNA sequencing was collected and used along with annotations to build a series of models predicting immune phenotypes from bulk RNAseq counts. First, important features were selected using multinomial ElasticNet[19] via the R package glmnet[43]. Next, these features were used as predictors for six different machine learning models. Three models were built via the R package XgBoost[20] using 10-fold cross validation alongside Bayesian parameter optimization[21] via the R package ParBayesianOptimization[44]. Three separate models were built with the R package ordinalForest[45]. Two models of the above six models were built considering both pathologists' predictions, two considering just pathologist-one and other two considering only pathologist-two. The classification accuracy of each algorithm for each group is available in **Supplemental Table 6**. Each algorithm uses the same cohort of 334 samples for training and 69 samples for testing. The algorithm with the best performance (determined with caret[46]) on the testing data was selected to be used going forward. More specific information is provided in **Supplemental Table 6**. Accuracy of the testing data for the model selected were visualized using ggplot2[47] and ggpubr[48].

Gastric cancer multiplex immunohistochemistry (mIHC) and flow cytometry. RNAseq count data was downloaded from Saito *et. al.*[16] along with flow cytometry and IHC outputs. These include 32 patients with gastric cancer in Tokyo[16]. These results were processed as previously described in Saito *et al.*[16] Using this resource, we used a MANOVA[49] to compare the IHC counts to determine if there were any significant differences between groups. Next, we used a Wilcox test[50] to assess the counts of each subset of IHC-marked cells to find significant differences. We performed pairwise comparisons of flow cytometry results using Wilcox tests[50]. Results were visualized using ggplot2[47] and ggpubr[48].

Determine resulting immune phenotype of each genomic alteration using RNAseq data. An R function was created which converts RNAseq count data to TPM using the R packages DESeq2[51] (to correct size factors) and DGE.obj.utils[52] (to convert to TPM), and then filters the data down to only the genes selected by ElasticNet[19]. The immune phenotype of each sample was predicted using XgBoost[20]. The proportion of high and low tumors for each cancer type were calculated. To analyze a genomic alteration, all mutant samples provided to the function are compiled, and a 1-proportion z-test is performed where the null hypothesis is that the proportion of hot samples will be equal to the population proportion within that cancer type and the alternative hypothesis is that the proportion of high samples is greater than the population proportion. If the null hypothesis is rejected ($p<0.05$), the genomic alteration is annotated RIGATonI-high. If not, a second 1-proportion z-test is performed where the null hypothesis is that the proportion of low samples will be equal to the population proportion and the alternative hypothesis is that the proportion of low samples is greater than the population proportion within that cancer type. If this null hypothesis is rejected ($p<0.05$), the genomic alteration is annotated RIGATonI-low. If not, the alteration is annotated “Unknown”.

Determining functional status of each genomic alteration using RNAseq data. The STRING[15] database’s protein action version 10.5 was downloaded. The list of protein actions is subset to include only actions on or by the protein of interest (POI). These actions are further filtered into two lists: an upstream protein list with only proteins that act to affect the expression of the POI, and a downstream protein list including all genes the POI activates, inhibits, or alters expression. The upstream gene list is used to model the RNAseq counts of the POI using modulators of the POI’s expression. The downstream gene list is used to model the RNAseq counts of the POI using downstream genes as indicators of its activity.

To model typical expression patterns of the POI, all samples with no alteration (control samples) in the gene of interest (GOI) are collected, and two generalized linear models are created over a Poisson distribution to predict the RNA counts of the POI/GOI. One model uses the upstream protein list as predictors, and another uses the downstream protein list. Both models predict the RNAseq counts of the POI.

Next, the RNAseq counts of the GOI within samples harboring mutations (mutant sample) predicted separately with each regression model. If, in either model, the expression of the GOI is lower than the lower bound of the 95% prediction interval (created by ciTools[53]), the mutant sample is annotated loss of function (LOF). On the other hand, if, in either model, the expression of the GOI is higher than the upper bound of the 95% prediction interval, the mutant sample is annotated gain-of-function (GOF). Falling outside the bounds of these prediction intervals indicates that the mutant sample’s GOI expression or activity is more different than that of a control sample than we would expect from random chance. If the mutant sample’s GOI expression falls within the bounds of both these prediction intervals, we can conclude that there is not enough evidence to indicate the mutation is causing abnormal expression or activity in that sample.

Next, all mutant samples provided are compiled, and a 2-proportion z-test is performed (null hypothesis z=0.5) comparing the proportion of LOF and GOF annotations. If the null hypothesis is rejected ($p<0.05$), the genomic alteration is annotated with the more frequent sample level annotation. If not, the alteration is annotated "Unknown."

Function annotation algorithm validation. OncoKB[17] provides a list of oncogenes with either targeting drugs or known oncogenic mutations. All alterations in these 1008 genes were analyzed in parallel. The results were filtered to only include GOF and LOF calls from the algorithm. Various alterations are called Unknown due to low sample count, confounding variables, or lack of known connections in STRING[15]. We did not include these samples in the pan-cancer analysis of TCGA, however users can elect to include them in their own analysis. Additionally, gene fusions were removed due to complexity of their calling. This left 1366 genomic alterations to investigate. We manually searched OncoKB[17] for information about each alteration and, if available, recorded the true function of the variant. Results were visualized with ggplot2[47] and ggpubr[48].

Building the R package. Functions were created with the R package devtools[54] which create an upstream and downstream gene list from STRING[17], determine the function of a group of samples using RNA expression data from bulk RNAseq, and the sample level immune phenotype using RNA expression data from bulk RNAseq. These functions are described in detail above. The R package along with relevant documentation is available at https://github.com/OSU-SRLab/RIGATONI.

Comprehensive analysis of genomic alterations in TCGA. All mutation information in TCGA was downloaded and compiled. A sample is said to have a copy number variation (CNV) if the total number of copies is ≥6 or <2. We also considered the sex of the patient in question if the gene of interest was on the X or Y chromosome, and we were considering a copy number loss. For male patients, genes on the X chromosome were said to be deleted if there were zero copies. For female patients, no genes on the Y chromosome were considered deleted. For any patients without biological sex available, we excluded them from the copy number analysis of genes on the X or Y chromosome. Next, all genomic alterations were analyzed in parallel through the RIGATonI R package described above. We ensured to analyze each alteration within each cancer type for both functional annotations and immune phenotyping. The population proportions used for immune phenotype were those of the cancer type being analyzed. Finally, we stored significant results from TCGA to be visualized by our online tool. The RIGATonI website is located at https://rigatoni.osc.edu/ and is managed by the Roychowdhury lab group and the Ohio Supercomputer Center. The website is built with the R package shiny[55], all graphs are visualized with ggplot2[47] and ggpubr[48]. The website input is a user provided GOI, and the website compiles gene level copy number, fusion, and simple single nucleotide variation results from the pan TCGA analysis. Next, we provide various visualizations to the user (quantiseqR[7] cell type proportions, RNAseq count data, and primary site prevalence) along with a table describing the different genomic alterations which were both functional and immunogenic within the GOI.

Text mining of PubMed abstracts to estimate novelty of RIGATONI TCGA output. PubMed abstracts containing the words "cancer" and "immunity" or "immunology" or "immune" since June 12th, 2010, were downloaded using the R package pubmed.mineR[56]. The function gene_atomization was used to perform text mining annotation of each gene mentioned. The RIGATonI results were extracted, and each gene was annotated with their frequency of appearance. Preprint publications were excluded from this analysis. 226,093 abstracts were analyzed. Results were visualized with ggplot2[47] and ggpubr[48].

ScRNAseq analysis of renal cell carcinoma. The data was analyzed using Seurat[57-60] Quality control measures were performed as follows: remove cells with <5x the standard deviation below median feature count, >5x the standard deviation above median feature count, <5x median total count, and <10% mitochondrial gene expression. We performed quality control steps for each sample individually. To mitigate experimental batch effect, we used harmony[61] and clustered using clustree[62]. To perform cell typing, we clustered all experiments together using UMAP with the Seurat[57-60] package. We then cell typed using the "kidney" tissue designation from ScType[63]. Any cells which were not able to be typed using the "kidney" designation were separated, clustered again using UMAP and Seurat[57-60], and re-typed using "Immune system" as the tissue of origin. Using copyKat[27], a scRNA-seq method to detect copy number alterations, we have identified 14q deleted tumors by using hematopoietic cells as a somatic control. We choose 14q deletion based on average copyKat score across chromosome 14q. If the score was less than zero on average, we determined that the sample harbored a 14q deletion. Additionally, we confirmed this by comparing the expression of all suspected 14q deleted tumor cells to all suspected wildtype tumor cells for each gene on chromosome 14q. For each gene, the 14q deleted tumors displayed significantly lower expression, Finally, using the FindAllMarkers function from Seurat[57-60], we investigated cell markers which were differentially expressed between 14q deleted and wildtype samples. These results were visualized using base Seurat[57-60] functions, ggplot2[47], and ggpubr[48].

Acknowledgements. The results published here are in whole or part based upon data generated by the TCGA Research Network: https://www.cancer.gov/tcga. All computational analyses were done on the Pitzer cluster at the Ohio Supercomputer Center (OSC) (https://www.osc.edu/). OSC also assisted in developing and hosting the RIGATonI website.

Author Contributions. RV conceived the idea for RIGATonI, developed the algorithms, wrote all R code for the project, wrote all Linux code along with ELH and ES, managed data, reviewed data analyses, and wrote/revised/edited the manuscript. ELH assisted with RIGATonI algorithm development, assisted with Linux coding, assisted with data management, and revised/edited the manuscript. LY assisted with RIGATonI algorithm development, assisted with data management, and revised/edited the manuscript. JWR, MRW, LS, and AP reviewed data analyses, revised/edited the manuscript. ES assisted with Linux coding, wrote all python code, and revised/edited the manuscript. EGB reviewed OncoKB and annotated the RIGATonI function algorithm's validation output. MC and NS enabled access and sequencing of the OSU-ORIEN dataset's RNAseq. WC and AF performed pathologist review of the OSU-ORIEN digital pathology images. SR conceived the idea for RIGATonI, reviewed data analyses, and wrote/revised/edited the manuscript.

References

1. Binnewies M, Roberts EW, Kersten K, Chan V, Fearon DF, Merad M, et al. Understanding the tumor immune microenvironment (TIME) for effective therapy. Nature Medicine. 2018;24(5):541-50.
2. Haslam A, Prasad V. Estimation of the Percentage of US Patients With Cancer Who Are Eligible for and Respond to Checkpoint Inhibitor Immunotherapy Drugs. JAMA network open. 2019;2(5):e192535.
3. Thorsson V, Gibbs DL, Brown SD, Wolf D, Bortone DS, Ou Yang TH, et al. The Immune Landscape of Cancer. Immunity. 2018;48(4):812-30.e14.
4. Nirmal AJ, Regan T, Shih BB, Hume DA, Sims AA-O, Freeman TC. Immune Cell Gene Signatures for Profiling the Microenvironment of Solid Tumors. (2326-6074 (Electronic)).
5. Thorsson V, Gibbs DL, Brown SD, Wolf D, Bortone DS, Ou Yang TH, et al. The Immune Landscape of Cancer. (1097-4180 (Electronic)).
6. Becht E, Giraldo NA, Lacroix L, Buttard B, Elarouci N, Petitprez F, et al. Estimating the population abundance of tissue-infiltrating immune and stromal cell populations using gene expression. Genome biology. 2016;17(1):218.
7. Finotello F, Mayer C, Plattner C, Laschober G, Rieder D, Hackl H, et al. Molecular and pharmacological modulators of the tumor immune contexture revealed by deconvolution of RNA-seq data. Genome Medicine. 2019;11(1):34.
8. Chen B, Khodadoust MS, Liu CL, Newman AM, Alizadeh AA. Profiling Tumor Infiltrating Immune Cells with CIBERSORT. Methods in molecular biology (Clifton, NJ). 2018;1711:243-59.
9. Charoentong P, Finotello F, Angelova M, Mayer C, Efremova M, Rieder D, et al. Pan-cancer Immunogenomic Analyses Reveal Genotype-Immunophenotype Relationships and Predictors of Response to Checkpoint Blockade. Cell Reports. 2017;18(1):248-62.
10. Li T, Fu J, Zeng Z, Cohen D, Li J, Chen Q, et al. TIMER2.0 for analysis of tumor-infiltrating immune cells. (1362-4962 (Electronic)).
11. Wang X, Chen L, Liu W, Zhang Y, Liu D, Zhou C, et al. TIMEDB: tumor immune micro-environment cell composition database with automatic analysis and interactive visualization. Nucleic acids research. 2023;51(D1):D1417-D24.
12. Motzer RJ, Robbins PB, Powles T, Albiges L, Haanen JB, Larkin J, et al. Avelumab plus axitinib versus sunitinib in advanced renal cell carcinoma: biomarker analysis of the phase 3 JAVELIN Renal 101 trial. Nature Medicine. 2020;26(11):1733-41.
13. Motzer RJ, Powles T, Atkins MB, Escudier B, McDermott DF, Alekseev BY, et al. Final Overall Survival and Molecular Analysis in IMmotion151, a Phase 3 Trial Comparing Atezolizumab Plus Bevacizumab vs Sunitinib in Patients With Previously Untreated Metastatic Renal Cell Carcinoma. (2374-2445 (Electronic)).
14. Saltz J, Gupta R, Hou L, Kurc T, Singh P, Nguyen V, et al. Spatial Organization and Molecular Correlation of Tumor-Infiltrating Lymphocytes Using Deep Learning on Pathology Images. (2211-1247 (Electronic)).
15. Szklarczyk D, Kirsch R, Koutrouli MA-O, Nastou KA-O, Mehryary FA-O, Hachilif R, et al. The STRING database in 2023: protein-protein association networks and functional enrichment analyses for any sequenced genome of interest. (1362-4962 (Electronic)).
16. Saito N, Sato Y, Abe H, Wada I, Kobayashi Y, Nagaoka K, et al. Selection of RNA-based evaluation methods for tumor microenvironment by comparing with histochemical and flow cytometric analyses in gastric cancer. Scientific Reports. 2022;12(1):8576.
17. Chakravarty D, Gao J, Phillips SM, Kundra R, Zhang H, Wang J, et al. OncoKB: A Precision Oncology Knowledge Base. LID - 10.1200/PO.17.00011 [doi] LID - PO.17.00011. (2473-4284 (Print)).
18. Yu ZA-O, Lv YA-O, Su CA-O, Lu WA-O, Zhang RA-OX, Li JA-O, et al. Integrative Single-Cell Analysis Reveals Transcriptional and Epigenetic Regulatory Features of Clear Cell Renal Cell Carcinoma. (1538-7445 (Electronic)).
19. Zou H, Hastie T. Regularization and Variable Selection Via the Elastic Net. Journal of the Royal Statistical Society Series B: Statistical Methodology. 2005;67(2):301-20.

20. Chen T, Guestrin C, editors. Xgboost: A scalable tree boosting system. Proceedings of the 22nd acm sigkdd international conference on knowledge discovery and data mining; 2016.
21. Jasper Snoek HL, Ryan P. Adams. Practical Bayesian Optimization of Machine Learning Algorithms. arXiv. 2012.
22. Voehringer D, Koschella M Fau - Pircher H, Pircher H. Lack of proliferative capacity of human effector and memory T cells expressing killer cell lectinlike receptor G1 (KLRG1). (0006-4971 (Print)).
23. Shin MS, Park H-J, Young J, Kang I. Implication of IL-7 receptor alpha chain expression by CD8+ T cells and its signature in defining biomarkers in aging. Immunity & Ageing. 2022;19(1):66.
24. Fenton SE, Saleiro DA-O, Platanias LC. Type I and II Interferons in the Anti-Tumor Immune Response. LID - 10.3390/cancers13051037 [doi] LID - 1037. (2072-6694 (Print)).
25. Wei SC, Duffy CR, Allison JP. Fundamental Mechanisms of Immune Checkpoint Blockade Therapy. Cancer Discovery. 2018;8(9):1069-86.
26. Huang J, Liu F, Li C, Liang X, Li C, Liu Y, et al. Role of CD47 in tumor immunity: a potential target for combination therapy. Scientific Reports. 2022;12(1):9803.
27. Gao R, Bai S, Henderson YC, Lin Y, Schalck A, Yan Y, et al. Delineating copy number and clonal substructure in human tumors from single-cell transcriptomes. Nature Biotechnology. 2021;39(5):599-608.
28. van Marcke C, Collard A, Vikkula M, Duhoux FP. Prevalence of pathogenic variants and variants of unknown significance in patients at high risk of breast cancer: A systematic review and meta-analysis of gene-panel data. Critical Reviews in Oncology/Hematology. 2018;132:138-44.
29. Thornton AM, Fang L, Lo A, McSharry M, Haan D, O'Brien C, et al. eVIP2: Expression-based variant impact phenotyping to predict the function of gene variants. PLOS Computational Biology. 2021;17(7):e1009132.
30. Alvarez MJ, Shen Y, Giorgi FM, Lachmann A, Ding BB, Ye BH, et al. Functional characterization of somatic mutations in cancer using network-based inference of protein activity. Nature genetics. 2016;48(8):838-47.
31. Lim Y, Choi S, Oh HJ, Kim C, Song S, Kim S, et al. Artificial intelligence-powered spatial analysis of tumor-infiltrating lymphocytes for prediction of prognosis in resected colon cancer. npj Precision Oncology. 2023;7(1):124.
32. Davenport T, Kalakota R. The potential for artificial intelligence in healthcare. (2514-6645 (Print)).
33. Kroeger N, Klatte T, Chamie K, Rao PN, Birkhäuser FD, Sonn GA, et al. Deletions of chromosomes 3p and 14q molecularly subclassify clear cell renal cell carcinoma. Cancer. 2013;119(8):1547-54.
34. Monzon FA, Alvarez K Fau - Peterson L, Peterson L Fau - Truong L, Truong L Fau - Amato RJ, Amato Rj Fau - Hernandez-McClain J, Hernandez-McClain J Fau - Tannir N, et al. Chromosome 14q loss defines a molecular subtype of clear-cell renal cell carcinoma associated with poor prognosis. (1530-0285 (Electronic)).
35. Martin Morgan SD, Marcel Ramos. GenomicDataCommons. 1.26.0 ed2023.
36. Zhang Y, Parmigiani G, Johnson WE. ComBat-seq: batch effect adjustment for RNA-seq count data. NAR Genomics and Bioinformatics. 2020;2(3):lqaa078.
37. Zarate S, Carroll A, Mahmoud M, Krasheninina O, Jun G, Salerno WJ, et al. Parliament2: Accurate structural variant calling at scale. GigaScience. 2020;9(12):giaa145.
38. Rausch T, Zichner T, Schlattl A, Stütz AM, Benes V, Korbel JO. DELLY: structural variant discovery by integrated paired-end and split-read analysis. Bioinformatics (Oxford, England). 2012;28(18):i333-i9.
39. Chen X, Schulz-Trieglaff O, Shaw R, Barnes B, Schlesinger F, Källberg M, et al. Manta: rapid detection of structural variants and indels for germline and cancer sequencing applications. Bioinformatics (Oxford, England). 2016;32(8):1220-2.
40. Fan X, Abbott TE, Larson D, Chen K. BreakDancer: Identification of Genomic Structural Variation from Paired-End Read Mapping. (1934-340X (Electronic)).
41. Lam HYK, Mu XJ, Stütz AM, Tanzer A, Cayting PD, Snyder M, et al. Nucleotide-resolution analysis of structural variants using BreakSeq and a breakpoint library. Nature Biotechnology. 2010;28(1):47-55.
42. Jeffares DA-O, Jolly C, Hoti M, Speed DA-O, Shaw L, Rallis C, et al. Transient structural variations have strong effects on quantitative traits and reproductive isolation in fission yeast. (2041-1723 (Electronic)).

43. Friedman J, Hastie T, Tibshirani R, Narasimhan B, Tay K, Simon N, et al. glmnet: Lasso and Elastic-Net Regularized Generalized Linear Models. cran2023.
44. Wilson S. ParBayesianOptimization: Parallel Bayesian Optimization of Hyperparameters. 1.2.6 ed2022.
45. Hornung R. Ordinal Forests. Journal of Classification. 2020;37(1):4-17.
46. Kuhn M. Building Predictive Models in R Using the caret Package. Journal of Statistical Software. 2008;28(5):1 - 26.
47. Wickham H. ggplot2: Elegant Graphics for Data Analysis: Springer-Verlag New York; 2016. Available from: https://ggplot2.tidyverse.org.
48. Kassambara A. ggpubr: 'ggplot2' Based Publication Ready Plots. 2023.
49. Smith H, Gnanadesikan R, Hughes JB. Multivariate Analysis of Variance (MANOVA). Biometrics. 1962;18(1):22-41.
50. Wilcoxon F. Individual comparisons by ranking methods. Biom. Bull., 1, 80–83. 1945.
51. Love MI, Huber W, Anders S. Moderated estimation of fold change and dispersion for RNA-seq data with DESeq2. Genome biology. 2014;15(12):550.
52. Law CW, Alhamdoosh MA-O, Su S, Dong X, Tian LA-O, Smyth GA-O, et al. RNA-seq analysis is easy as 1-2-3 with limma, Glimma and edgeR. LID - ISCB Comm J-1408 [pii] LID - 10.12688/f1000research.9005.3 [doi]. (2046-1402 (Electronic)).
53. John Haman MA, Institute for Defense Analyses. ciTools: Confidence or Prediction Intervals, Quantiles, and Probabilities for Statistical Models. 2020.
54. Hadley Wickham JH, Winston Chang, Jennifer Bryan, RStudio. devtools: Tools to Make Developing R Packages Easier. 2.4.5 ed2022.
55. Winston Chang JC, JJ Allaire, Carson Sievert, Barret Schloerke, Yihui Xie, Jeff Allen, Jonathan McPherson, Alan Dipert, Barbara Borges, Posit Software, PBC, jQuery Foundation, jQuery contributors, jQuery UI contributors, Mark Otto, Jacob Thornton, Bootstrap contributors, Twitter, Inc, Prem Nawaz Khan, Victor Tsaran, Dennis Lembree, Srinivasu Chakravarthula, Cathy O'Connor, PayPal, Inc, Stefan Petre, Andrew Rowls, Brian Reavis, Salmen Bejaoui, Denis Ineshin, Sami Samhuri, SpryMedia Limited, John Fraser, John Gruber, Ivan Sagalaev, R Core Team. shiny: Web Application Framework for R. 1.8.0 ed2023.
56. Rani J, Shah Ab Fau - Ramachandran S, Ramachandran S. pubmed.mineR: an R package with text-mining algorithms to analyse PubMed abstracts. (0973-7138 (Electronic)).
57. Hao Y, Hao S, Andersen-Nissen E, Mauck WM, 3rd, Zheng S, Butler A, et al. Integrated analysis of multimodal single-cell data. Cell. 2021;184(13):3573-87.e29.
58. Stuart T, Butler A, Hoffman P, Hafemeister C, Papalexi E, Mauck WM, 3rd, et al. Comprehensive Integration of Single-Cell Data. Cell. 2019;177(7):1888-902.e21.
59. Butler A, Hoffman P, Smibert P, Papalexi E, Satija R. Integrating single-cell transcriptomic data across different conditions, technologies, and species. Nature Biotechnology. 2018;36(5):411-20.
60. Satija R, Farrell JA, Gennert D, Schier AF, Regev A. Spatial reconstruction of single-cell gene expression data. Nature Biotechnology. 2015;33(5):495-502.
61. Korsunsky I, Millard N, Fan J, Slowikowski K, Zhang F, Wei K, et al. Fast, sensitive and accurate integration of single-cell data with Harmony. Nature Methods. 2019;16(12):1289-96.
62. Zappia L, Oshlack A. Clustering trees: a visualization for evaluating clusterings at multiple resolutions. GigaScience. 2018;7(7):giy083.
63. Ianevski A, Giri AK, Aittokallio T. Fully-automated and ultra-fast cell-type identification using specific marker combinations from single-cell transcriptomic data. Nature Communications. 2022;13(1):1246.

Supplemental Software File for RIGATonI

Raven Vella, Emily Hoskins, and Eric Samorodnitsky

Contents

Libraries

Load necessary libraries:

```
library(maftools)
library(vcfR)
library(foreach)
library(TCGAutils)
library(GenomicDataCommons)
library(glmnet)
library(ordinalForest)
library(xgboost)
library(ParBayesianOptimization)
library(data.table)
library(readr)
library(sva)
library(pROC)
library(caret)
library(DGEobj.utils)
library(MASS)
library(ordinalForest)
library(ciTools)
library(biomaRt)
library(DESeq2)
library(DGEobj.utils)
library(TCGAutils)
library(pubmed.mineR)
library(ggrepel)
library(scales)
library(ggplot2)
library(ggpubr)
library(Seurat)
library(harmony)
library(clustree)
library(dplyr)
library(openxlsx)
library(HGNChelper)
library(parallel)
library(copykat)
```

Part 1: Process simple nucleotide variations (SNVs), copy number variations (CNVs), Structural Variants (SVs), and gene fusion outputs from TCGA

In order to run RIGATONI on an entire database, you need to have all the fusion, SNV, CNV, and SV outputs from said database in list format. I used the outputs directly from TCGA for the SNV, CNV, and Fusions callers. I processed the whole genome samples using parliament2 for SVs myself. Shown below are the steps for collating and organizing the SNVs, CNVs, and fusions.

1.1: Download SNV, CNV, and fusion outputs from TCGA

First, I downloaded the SNV, CNV, and fusion outputs from TCGA. I created manifests using the GDC database and downloaded the files as shown below.

```
# read in manifest file and call it mani
mani = read.table('<manifest name>', sep = "\t", header = T)

# save the token as an environment variable in R
Sys.setenv(GDC_TOKEN = readLines('<token file name>'))

# set cache and gdc_client software paths
options(gdc_client = "<client location>")
gdc_set_cache(directory = '<output location>')

# download each item in the manifest, if you do not have access to the item,
# skip it and move on
fnames = lapply(mani$id, function(x)

  tryCatch(
    suppressMessages(

      gdcdata(
        x,
        progress = FALSE,
        access_method = "api",
        use_cached = FALSE,
        token = gdc_token()
      )

    ),

    error = function(e) {

      message(paste0('No access to ', x))
      return(NA)
    }
  ))
```

1.2: Process SNV outputs into a list

```
# read in list of genes for which we have expresssion data
genes = readLines('<TCGA-genes>')

# go to directory where maf files are located
setwd("<maf location>")

# list all directories there
dir = list.dirs()

# filter out the directory "."
dir = dir[2:length(dir)]

l <- foreach(x = dir) %do% {
  # enter each directory
  setwd(paste0(x))

  if (length(list.files(pattern = "*.maf.gz")) > 0) {
    # if there is a maf file there, read it in. If not, skip it
    y <-
      tryCatch(

        maftools::read.maf(list.files(pattern = "*.maf.gz")),

        error = function(e) {
          return(NA)
        }

      )

    return(y)
  }

  # go back to the initial directory
  setwd("<maf location>")
}

# remove NA entries from the list
l = l[!(is.na(l))]

# go to the directory where the list should be stored
setwd('<final location>')

# save the initial list in case R crashes
# (this is a large file, and functions on it sometimes exceed available ram)
saveRDS(l, 'MafsTogether.RDS')

# stack all the individual files together into one data frame
```

```
l = do.call(rbind, l)

# split the dataframe based on the gene symbol of the alteration call
l = split(l, l$Hugo_Symbol)

# remove entries for which we do not have expression data
l = l[names(l) %in% genes]

#Save this final list
saveRDS(l, 'MafsTogether2.RDS')
```

1.3: Process fusion outputs into a list

```
#Go to directory where fusion files are located
setwd("<fusions location>")

# list directories
dir = list.dirs()

# remove directories which are "logs" of the downloads
dir = dir[!(grepl("logs", dir))]

# remove the directory "."
dir = dir[2:length(dir)]

# initiate empty vector "names"
names = c()

fus <- foreach(x = dir) %do% {
  # for each directopry, go to the fusion parent directory
  setwd("<fusions location>")

  # enter the directory of interest
  setwd(x)

  # if there are no fusion outputs (files ending in .tsv),
  # skip this directory
  if (length(list.files(pattern = "*.tsv")) > 0) {

    for (f in list.files(pattern = "*.tsv")) {
      # read the fusion output file and store it as a dataframe "y"
      y <- read.delim(f,
                      sep = "\t",
                      header = T,
                      check.names = FALSE)

      # If y has no rows (no fusion calls) skip it.
      if (nrow(y) > 0) {

        if (colnames(y)[2] == 'gene2') {
```

```
      # if y is an ARRIBA output,
      # filter y to include only the columns listed
      y = y[, c("#gene1", "gene2", "breakpoint1", "breakpoint2")]

    } else {

      # if y is an STAR fusion output,
      # filter y to include only the columns listed
      y = y[, c("LeftGene",
                "RightGene",
                "LeftBreakpoint",
                "RightBreakpoint")]

      # remove "^" characters from the gene names
      y$LeftGene = gsub("\\^..*", "", y$LeftGene)
      y$RightGene = gsub("\\^..*", "", y$RightGene)

    }

    # rename the columns of y
    colnames(y) = c("gene1", "gene2", "breakpoint1", "breakpoint2")

    # create new column called chr1 made up of the chromosome
    # from breakpoint 1
    y$chr1 = unlist(lapply(y$breakpoint1, function(x) {
      return(strsplit(x, ":")[[1]][1])
    }))

    # edit breakpoint 1 to only be location on the chromosome
    y$breakpoint1 = unlist(lapply(y$breakpoint1, function(x) {
      return(as.numeric(strsplit(x, ":")[[1]][2]))
    }))

    # create new column called chr2 made up of the chromosome
    # from breakpoint 2
    y$chr2 = unlist(lapply(y$breakpoint2, function(x) {
      return(strsplit(x, ":")[[1]][1])
    }))

    # edit breakpoint 2 to only be location on the chromosome
    y$breakpoint2 = unlist(lapply(y$breakpoint2, function(x) {
      return(as.numeric(strsplit(x, ":")[[1]][2]))
    }))

    # add sample id using file name
    y$SampleID = f

    # convert file name to case ID
    y$CaseID = UUIDtoUUID(filenameToBarcode(f)[1, 2],
                          to_type = 'case_id')[1, 2]

    # append filename to the names vector
    names = c(names, f)
```

```
        # remove rows from y which contain genes for which we have
        # no expression data
        out = unlist(lapply(1:nrow(y), function(r) {

          if (y[r, 1] %in% genes && y[r, 2] %in% genes) {

            return(T)

          } else {

            return(F)

          }
        }))

        y = y[out,]

        # only return y if the final dataframe has more than 0 rows
        if (nrow(y) > 0) {

          return(y)

        } else {

          return(NA)

        }
      } else {
        return(NA)
      }
    }
  } else {

    #return to the parent directory
    setwd("<fusions location>")

  }
}

# set names of the fus list to the file names
names(fus) = names

# remove skipped entries from the fus list
fus = fus[!(is.na(fus))]

# return to the directory where the list should be stored
setwd('<final location>')

# stack the fusion outputs together into a dataframe
fus = do.call(rbind, fus)

# save the fusion dataframe in case of ram issues
saveRDS(fus, 'FusTogether.RDS')
```

```
# double check that all genes have accompanying expression data
fus = fus[fus$gene1 %in% genes,]
fus = fus[fus$gene2 %in% genes,]

# create a new column "Combo" that names each fusion using the two genes
fus$Combo = paste0(fus$gene1, "-", fus$gene2)

# create new list by splitting the fusion dataframe by fusion name
fus2 = split(fus, fus$Combo)

for (x in 1:length(fus2)) {
  # for each fusion, save the dataframe as new
  new = fus2[[x]]

  # split new into subdataframe based on the case ID
  new = split(new, new$CaseID)

  for (y in 1:length(new)) {
    # for each case ID, save the dataframe as newnew
    newnew = new[[y]]

    if (nrow(newnew) > 1) {
      # if newnew has more than 1 row, this means there are multiple calls
      # for the same fusion create 2 matrices comparing each call's
      # first and second breakpoint
      br1 = as.data.frame(expand.grid(newnew$breakpoint1, newnew$breakpoint1))
      br2 = as.data.frame(expand.grid(newnew$breakpoint2, newnew$breakpoint2))

      # create new columns with the difference of these breakpoints
      br1$dif = abs(br1[, 1] - br1[, 2])
      br2$dif = abs(br2[, 1] - br2[, 2])

      # remove rows from each matrix where the breakpoints are
      # at most 100 bp apart
      br1 = br1[br1$dif >= 100 ||
                  br1$dif == 0,]
      br2 = br2[br2$dif >= 100 ||
                  br2$dif == 0,]

      # take only the first column of each dataframe
      br1s = br1[, 1]
      br2s = br2[, 1]

      # subset newenw to only contain these filtered breakpoints
      newnew = newnew[newnew$breakpoint1 %in% br1s &&
                        newnew$breakpoint2 %in% br2s,]

    }
    # replace the previous version of newnew with the filtered version
    new[[y]] = newnew

  }
  # stack the filtered versions together to created filtered
  # dataframe for the fusion
  new = do.call(rbind, new)

  # replace previous fusion dataframe with filtered version
  fus2[[x]] = new

}

# stack filtered fusions on top of each other to create new dataframe fus2
fus2 = do.call(rbind, fus2)

# add the chromosome of each breakpoint back into the breakpoint columns
fus2$breakpoint1 = paste0(fus2$chr1, ":", fus2$breakpoint1)
fus2$breakpoint2 = paste0(fus2$chr2, ":", fus2$breakpoint2)

# remove the now redundant breakpoint columns
fus2 = fus2[, c('gene1',
                'gene2',
                'breakpoint1',
                'breakpoint2',
                'CaseID',
                'Combo')]

# split by the fusion name
fus2 = split(fus2, fus2$Combo)

# return to where you are storing the lists
setwd('<final location>')

# save the new fusion list
saveRDS(fus2, 'FusTogether2.RDS')
```

1.4: Process CNV outputs into a list

```
# go to where the copy number outputs are stored
setwd('<copy number location>')

# list files in this directory
ls = list.files()

# go through the list of files individually
ls1 = lapply(ls, function(x) {

  # read the file in and store it as dataframe cn
  cn = read.delim(x, header = T, sep = '\t')

  # remove empty rows
  cn = cn[!(is.na(cn$copy_number)),]

  # create list of rows which have either CNV gain or loss
  use = unlist(lapply(1:nrow(cn), function(r) {
    if (cn$copy_number[r] < 2 ||
        cn$copy_number[r] >= 6 &&
        cn$gene_name[r] %in% genes) {

      return(T)

    } else {

      return(F)

    }
  }))

  # subset the dataframe to only contain those rows identified above
  cn = cn[use,]

  # include only columns listed
  cn = cn[, c('gene_name', 'start', 'end', "copy_number")]

  # edit copy number column to contain the string "CNV"
  cn$copy_number = paste0("CNV: ", cn$copy_number)

  # add a case ID column using the file name
  cn$CaseID = UUIDtoUUID(gsub("\\..*", "", x))[1, 2]

  return(cn)
})

# stack the entries in the list on top of each other in dataframe ls1
ls1 = do.call(rbind, ls1)

# split the new ls1 datafream by gene_name
ls1 = split(ls1, ls1$gene_name)

# go to desired directory
setwd('<final location>')

# store the resulting list
saveRDS(ls, "CopyTogether2.RDS")
```

1.5: Get parliament2 results for whole genome samples

First you need to get the parliament2 image from dnanexus

```
singularity pull docker://dnanexus/parliament2
```

Next I wrote the following script to run parliament2 on all whole genome samples in parallel

```
#!/bin/sh
#SBATCH --account=PAS0854
#SBATCH --time=96:00:00
```

```
#SBATCH --ntasks=3

Help()
{
# Display Help
echo "Run the SV calls on all TCGA bams"
echo
echo "Syntax: GridssAll.sh [-h|o|r|p|u|k]"
echo "options:"
echo "h     Print help."
echo "o     Output path"
echo "r     Reference Path"
echo "p     Paired bam file name"
echo "u     Username"
echo "k     Token path"
echo
}

while getopts "h:o:r:p:u:k:" option; do
case $option in
h) # display Help
Help
exit;;
u) #store the user name
user=$OPTARG
;;
o) #store the output path
out=$OPTARG
;;
r) #store reference path
ref=$OPTARG
;;
p) #store bam UUID name
pair=$OPTARG
;;
k) #store token path
token=$OPTARG
;;
\?) #invalid option
echo 'Error: Invalid input, please use -h for help'
exit;;
esac
done

# go line by line through the bam list
while IFS= read line;
do
  cond=`squeue -u $user | wc -l`
  # check that we are not going to exceed 1000 jobs
  while [ $cond -ge 990 ];
  do
    # if we are, wait 30 min and check again
    echo "Waiting: SV called";
    sleep 10m;
    cond=`squeue -u $user | wc -l`;
  done;
  sbatch runCallers.sh \
  -t `echo $line` -r $ref -o $out -k $token;
done < $pair
```

To run this script, I would use the line of code below (replaceing with the appropriate file paths).

```
sbatch getSVresults.sh -u <osc_username> -o <output_path> -r <reference_path> \
-p <uuid_list> -t <token_path>
```

The runCallers.sh script is shown below.

```
#!/bin/sh
#SBATCH --account=PAS0854
#SBATCH --time=12:00:00
#SBATCH --ntasks=5

Help()
{
# Display Help
echo "Run the callers on a bam"
echo
echo "Syntax: runCallers.sh [-h|o|r|t|n|s|k]"
echo "options:"
echo "h     Print help."
echo "o     Output path"
echo "r     Reference Path"
echo "t     Tumor bam file name"
echo "s     Somatic bam file name"
echo "k     Token file name"
echo
}

while getopts "h:o:r:t:n:s:k:" option; do
case $option in
h) # display Help
Help
exit;;
o) #store the output path
out=$OPTARG
;;
r) #store reference path
ref=$OPTARG
;;
t) #store paired bam file name
tumor=$OPTARG
;;
k) #store token
token=$OPTARG
;;
\?) #invalid option
```

```
echo 'Error: Invalid input, please use -h for help'
exit;;
esac
done

#navigate to the ouput location
cd $out

#create working directory for the tumor sample
mkdir $tumor.working/

#temporarily copy the reference files to the working directory
cp $ref.gz $tumor.working/$ref.gz
cp $ref.fai $tumor.working/$ref.fai

#enter the working directory
cd $tumor.working

#make a new directory for the output
mkdir output/

#store current directory in variable $dir
dir=$(pwd)

#store token in variable $token
token=$(<$token)

#download the tumor bam from GDC
curl -o ./$tumor.bam --header "X-Auth-Token: $token" \
'https://api.gdc.cancer.gov/data/'$tumor

#load samtools
module load samtools

#index the bam
samtools index ./$tumor.bam ./$tumor.bam.bai

#run parliament2 with manta, delly, breakdancer, and breakseq
singularity run --bind $dir:/home/dnanexus/in,$dir/output:parliament2_latest.sif \
--bam $tumor.bam \
--bai $tumor.bam.bai --fai $ref.fai -r $ref.gz \
--manta --delly_deletion --delly_insertion --delly_inversion \
--delly_duplication --breakdancer --breakseq --genotype

#move the output files to the initial out directory
mv ./output/$tumor.survivor_sorted.vcf $out/$tumor.survivor_sorted.vcf

#go back to the out directory
cd $out

#remove the temporary files
rm -r $tumor.working/
```

1.6: Process SV outputs into a list

```
#go to the directory where the SV results are stored.
setwd('<structural variant location>')

#create a list of the .vcf files in this directory
lsv = list.files(pattern = "*.annotated")

lsv <- foreach(i = lsv) %do% {
  #for each vcf file, attempt to read it in, if an error is returned, skip it
  a <-
    tryCatch(

      vcfR::read.vcfR(i, verbose = FALSE),

      error = function(e) {

        return(NA)

      }
    )

  return(a)
}

#set the names of the list to the names of the files
names(lsv) = list.files(pattern = "*.annotated")

#removed skipped values
lsv = lsv[!(is.na(lsv))]

newlsv <- foreach(i = 1:length(lsv)) %do% {
  #for each entry in the list, store the name of the entry as nam
  nam = names(lsv)[i]

  #next store the object as i
  i = lsv[[i]]

  #turn i into a tidy
  i = vcfR::vcfR2tidy(i)

  #extract the data frame
  i = i$fix

  #remove entries that do not effect a gene
  i = i[!(is.na(i$GENE)),]

  #add a column with the name of the file
  i$SampleID = nam

  return(i)
}
```

```r
#create list of objects in the list with no SV results
out = unlist(lapply(newlsv, function(x) {

  if (nrow(x) == 0) {

    return(F)

  } else {

    return(T)

  }
}))

#remove the entries with no SV results
sv = newlsv[out]

#go to desired directory
setwd('<final location>')

#store this list in case of ram issues
saveRDS(sv, "SVTogether.RDS")

#create biomart object with ensembl names
mart <- useDataset("hsapiens_gene_ensembl", useMart("ensembl"))

#create empty list sv2
sv2 = list()

for (x in sv) {
  #for each dataframe in sv, store it as a new variable dat
  dat = x

  #store the gene names in a vector called rid
  rid = dat$GENE

  #remove genes that are not listed as mRNA transcripts
  rid = rid[grepl('NM', rid)]

  #remove isoform information
  rid = gsub("\\..*", "", rid)

  #create new dataframe with entries in rid along with corresponding hgnc symbols names
  refseq_mapping <-
    biomaRt::getBM(
      attributes = c("refseq_mrna", "hgnc_symbol"),
      filters = "refseq_mrna",
      values = rid,
      mart = mart
    )
  #create a new data frame from gene information in dat without isoform infromation
  rid2 = gsub("\\..*", "", dat$GENE)
  #replace the entries in the new list with the hgnc symbols
  for (i in 1:nrow(refseq_mapping)) {
    rid2 = replace(rid2, which(rid2 == refseq_mapping[i, 1]), refseq_mapping[i, 2])
  }

  #replace ensemble symbols in dat with the hgnc symbols
  dat$GENE = rid2

  #remove genes for which we have no expression information
  dat = dat[dat$GENE %in% genes,]

  #fine case ID using the file name
  caseID = UUIDtoUUID(gsub("\\..*", "", dat$SampleID[1]),
                      to_type = 'case_id')[1, 2]

  #subset dat to only include columns listed
  dat = dat[, c('POS', 'END', 'GENE', 'ALT')]

  #add column with the case ID information
  dat$CaseID = caseID

  #append dat to the new list sv2
  sv2[[length(sv2) + 1]] = dat

}

#stack dataframes in sv2 on top of eachother
sv2 = do.call(rbind, sv2)

#split the sv2 dataframe into a list by gene
sv2 = split(sv2, sv2$GENE)

#navigate to desired directory
setwd('<final location')

#save sv2
saveRDS(sv2, 'SVTogether2.RDS')
```

Part 2: Batch Correction on TCGA expression files

In order to analyze the RNA seq data all together from TCGA, batch correction was performed by institution. First all RNA expression files must be downloaded. To do this, I went to GDC, created a manifest and downloaded them as shown below.

```
#read in manifest file and call it mani
mani = read.table('<manifest name>', sep = "\t", header = T)

#save the token as an environment variable in R
Sys.setenv(GDC_TOKEN = readLines('<token file name>'))

#set cache and gdc_client software paths
options(gdc_client = "<client location>")
```

```r
gdc_set_cache(directory = '<output location>')

# download each item in the manifest, if you do not
# have access to the item, skip it and move on
fnames = lapply(mani$id, function(x)
  tryCatch(

    suppressMessages(

      gdcdata(
        x,
        progress = FALSE,
        access_method = "api",
        use_cached = FALSE,
        token = gdc_token()
      )

    ),

    error = function(e) {

      message(paste0('No access to ', x))

      return(NA)

    }
  ))
```

Next, you need to unpack all the directories.

```
cd <path/to/directories>
find . -maxdepth 1 -exec mv {} .. \;
```

Next, I ran combat-seq from SVA as shown below.

```r
#Go to the location of the RNA seq data
setwd("<location of gene expression files>")

#get list of the counts files
fl = list.files(pattern = '*star_gene_counts.tsv')

#read in each file
counts = lapply(fl, function(x) {

  #read in the tsv file skipping the header line
  test = as.data.frame(read_tsv(x, skip = c(1)))

  #select only the gene name and counts columns
  test = test[, c(2, 4)]

  #remove rows with NA entries
  test = test[!(is.na(test$gene_name)), ]
  test = test[!(is.na(test$unstranded)), ]
  #make sure the counts column is numeric
  test$unstranded = as.numeric(as.character(test$unstranded))

  #aggregate reads together by gene name
  test = aggregate(test$unstranded, list(test$gene_name), sum)

  #save the gene names
  names = test$Group.1

  #remove the first column of gene names
  test = test[,-1]

  #convert to a dataframe
  test = as.data.frame(test)

  #add rownames (gene names) back to the file
  rownames(test) = names

  #return data frame
  return(test)
})

#combine counts side by side
counts = do.call(cbind, counts)

#get the barcodes from the file names
fl = filenameToBarcode(fl)
fl = fl$aliquots.submitter_id

#set the column names to the barcodes
colnames(counts) = fl

setwd("<final location>")

#save the row names (gene names)
fConn = file('Gene_Names.txt')
writeLines(rownames(counts), fConn)
close(fConn)

#Create your batches
bat = unlist(lapply(fl, function(x) {

  #the center is the 7th entry in the barcode
  return(strsplit(x, "-")[[1]][7])

}))

#run combat
adj_counts = ComBat_seq(counts, bat)

#save the results
setwd('<final location>')
lapply(1:ncol(adj_counts), function(x){
  write.csv(
    adj_counts[, x],
    file = paste0(colnames(adj_counts)[x], "_batch_corrected.csv"),
    row.names = T)

  })
```

Part 3: Build hot/cold detection algorithm

3.1: Preprocessing input data

First I combine the pathologist annotations and gather the paired RNA for each sample.

```
#read in the data
tru1 = readxl::read_xlsx('<path 1>', sheet = 1)
tru1 = as.data.frame(tru1)
tru2 = readxl::read_xlsx('<path 2>', sheet = 1)
tru2 = as.data.frame(tru2)
colnames(tru2) = colnames(tru1)

#remove first three columns of notes
tru2 = tru2[-c(1:3),]

#remove question marks from pathologist annotations
tru2$`Lymphocyte Annot.` = gsub("\\?", "", tru2$`Lymphocyte Annot.`)

#convert annotations to ordered values
tru1$`Lymphocyte Annot.` = ifelse(tru1$`Lymphocyte Annot.` == 'hot',
                                  2,
                                  ifelse(tru1$`Lymphocyte Annot.` == 'cold',
                                         0,
                                         1))
tru2$`Lymphocyte Annot.` = ifelse(tru2$`Lymphocyte Annot.` == 'hot',
                                  2,
                                  ifelse(tru2$`Lymphocyte Annot.` == 'cold',
                                         0,
                                         1))
#combine the two data tables
tru = as.data.frame(cbind(tru1, tru2$`Lymphocyte Annot.`))
colnames(tru)[ncol(tru)] = 'path1'
colnames(tru)[ncol(tru) - 2] = 'path2'

#remove initial data
rm(tru1)
rm(tru2)

#combine the pathologist annotations into one final column
tru$true = sum(tru$path1, tru$path1)

#read in RNA
rna = readRDS('<batch adjusted RNA counts>')
```

```
#Make sure the RNA columns match the annotation rows
rna = as.data.frame(rna)
rna = rna[, colnames(rna) %in% tru$`RNASeq SLID`]
rna = rna[, match(tru$`RNASeq SLID`, colnames(rna))]

#create new column with final annotations using both pathologist reviews
tru$group = ifelse(tru$true == 0,
                   'low',
                   ifelse(tru$true == 4,
                          'high',
                          'medium'))

#save tru for future use
saveRDS(tru, '<true annotations>')

#create balanced list of samples to use for training (80% of high, medium, and low)
train = split(tru, as.character(tru$group))
train = lapply(train, function(x) {

  ids = sample(1:nrow(x), nrow(x) * .8)

  return(x[ids,])
})

#record the training data list for future use
train = do.call(rbind, train)
saveRDS(train, 'training.ORIEN.RDS')

# Prior to training the algorithm, some normalization steps are required.
# First we filter the genes available to include those in the validation data sets.
# Next we normalize the data according to size factors and convert to TPM.
# These steps ensure the algorithm will work consistently across data sources

#load gene lengths
geneLen = readRDS('<gene lengths>')

#filter to inlcude only genes present across different datasets
genes_TCGA = readLines("<Gene Names TCGA>")
genes_IHC = readRDS('<raw.data gastric_IHC.RDS>')
genes_IHC = genes_IHC$TPM
genes_IHC = rownames(genes_IHC)
genes = intersect(intersect(geneLen$Gene_Symbol, genes_TCGA), genes_IHC)

#filter RNA to only include genes in the IHC dataset, ORIEN, and TCGA
rna = rna[rownames(rna) %in% genes,]

#create new dataframe with annotations from tru
coldata = as.character(tru$true)
coldata = as.data.frame(coldata)
colnames(coldata) = c('condition')
coldata$condition = as.factor(coldata$condition)

#create count data frame with size normalization
```

```r
t_rna = DESeqDataSetFromMatrix(countData = rna,
                               colData = coldata,
                               design = ~ condition)
t_rna <- estimateSizeFactors(t_rna)
t_rna = counts(t_rna, normalized = T)

#make sure the gene lengths are in the same order as the rownames from t_rna
geneLen = readRDS('/fs/ess/PAS0854/Raven/immune-genomics/ValidateRiga/GeneLengths.RDS')
geneLen = geneLen[geneLen$Gene_Symbol %in% rownames(t_rna),]
geneLen = geneLen[match(rownames(t_rna), geneLen$Gene_Symbol),]

#convert the normliazed RNA to counts
t_rna = convertCounts(as.matrix(t_rna), 'TPM', geneLen$Length)
t_rna = t(t_rna)

#save the RNA for future use
saveRDS(t_rna, 'RNA.Orien.All.RDS')
```

3.2 ElasticNet feature selection

Next we will select features for our machine learning algorithm using elastic net. This was done for all 3 types of annotations (pathologist 1, pathologist 2, and the combination.)

```r
#create new variable x.vars using t_rna
x.vars = t_rna

# the following is based on the tutorial found at "https://rpubs.com/jmkelly91/881590"
# first we will tune the parameter alpha
models <- list()

for (i in 0:20) {
  # print alpha
  name <- paste0("alpha", i / 20)

  # create model for the given alpha value
  models[[name]] <-
    cv.glmnet(as.matrix(x.vars[train$`RNASeq SLID`,]),
              train$true,
              family = "poisson",
              alpha = i / 20)
}

# store the error in a results table for each alpha
results <- data.frame()

for (i in 0:20) {
  # print alpha
  name <- paste0("alpha", i / 20)

  # record the predicted values for each sample
  test = cbind(
    train$true,
    predict(
      models[[name]],
      lambda = 'lambda.min',
      newx = as.matrix(x.vars[train$`RNASeq SLID`, ]),
      type = 'response'
    )
  )
  test = as.data.frame(test)
  colnames(test) = c('Actual', 'Predicted')
  true = test$Actual
  predicted = test$Predicted

  #calculate the mean squared error for each sample
  mse <- mean((true - predicted) ^ 2)

  ## Store the results
  temp <- data.frame(alpha = i / 20,
                     mse = mse,
                     name = name)

  results <- rbind(results, temp)
}

plot(results$alpha, results$mse)

#choose model with lowest MSE
best_model = models[["alpha0"]]
best_alpha = 0.00

#record the lambda min for the model selected
best_lam <- best_model$lambda.min

#build the new model using the alpha and lambda values selected
lasso_best <-
  glmnet(
    as.matrix(x.vars[train$`RNASeq SLID`, ]),
    train$true,
    family = "poisson" ,
    alpha = best_alpha,
    lambda = best_lam
  )

# extract the features from from the best lasso model.
mat = coef(lasso_best)
mat = as.matrix(mat)
mat = mat[mat[, 1] != 0,]
mat = mat[-1]
mat = mat[order(-abs(mat))]
genes = names(mat[abs(mat) > .01])

#record the genes for future use
saveRDS(genes, 'genes.RDS')
```

3.3 OrdinalForest Machine learning

After all the genes are collected, we built two kinds of machine learning models for each annotation: OrdinalForest and XgBoost. The process for OrdinalForest is shown below

```r
#filter t_rna to only include genes from the elastic net output
t_rna = t_rna[, colnames(t_rna) %in% genes]

#create new variable x.vars with t_rna
x.vars = as.data.frame(t_rna)

#add the tru annotation of interest to the data
x.vars$Status = tru$group_of_interest

#change the factor levels
x.vars$Status = factor(x.vars$Status,
                       levels = c('low', 'medium', 'high'))

#run orindal forest
m <-
  ordfor(depvar = 'Status',
         data = x.vars[train$`RNASeq SLID`,],
         perffunction = "probability")

#save the model
saveRDS(m, 'model.RDS')

#create a list of testing samples excluding the training data
test = tru$`RNASeq SLID`[!(tru$`RNASeq SLID` %in% train$`RNASeq SLID`)]

#filter x.vars to only include the testing data set
test = x.vars[test, ]

#remove the status for evaluation with the new ordinal forest model
test = test[, colnames(test) != 'Status']

#predict the new values
pred = predict(m, newdata = test, type = 'class')

#combine the prediction with the true values for the testing data
final <-
  cbind(as.character(pred$ypred),
        tru[match(rownames(test), tru$`RNASeq SLID`), 'group'])
colnames(final) = c('Predicted', 'Actual')
final = as.data.frame(final)

#make sure the factor levels are in the desired order
final$Predicted = factor(final$Predicted,
                         levels = c('high', 'medium', 'low'))
final$Actual = factor(final$Actual,
                      levels = c('high', 'medium', 'low'))

#evaluate with a confusion matrix from caret
confusionMatrix(final$Predicted, final$Actual)
```

3.4 XgBoost machine learning

Below is shown the process for machine learning with XgBoost. This is based on a tutorial found at "https://www.r-bloggers.com/2022/01/using-bayesian-optimisation-to-tune-a-xgboost-model-in-r/"

```
#filter t_rna to only include genes from the elastic net output
t_rna = t_rna[, colnames(t_rna) %in% genes]

#create the training data as an xgboost matrix
dtrain <- xgb.DMatrix(t_rna[train$`RNASeq SLID`,],
                      label = train$group_of_interest)

# build the objective function for the optimization
obj_func <-
  function(eta,
           max_depth,
           min_child_weight,
           subsample,
           lambda,
           alpha) {

    #create a param list
    param <- list(
      # Hyter parameters
      eta = eta,
      max_depth = max_depth,
      min_child_weight = min_child_weight,
      subsample = subsample,
      lambda = lambda,
      alpha = alpha,
      booster = "gbtree",
      objective = 'multi:softmax',
      eval_metric = "auc",
      num_class = 3
    )

    # run a cross validated xgboost with the set parameters
    xgbcv <- xgboost::xgb.cv(
      params = param,
      data = dtrain,
      nrounds = 500,
      nfold = 5,
      stratified = T,
      early_stopping_rounds = 10,
      verbose = 0,
      maximize = TRUE
    )

    lst <- list(
      # First argument must be named as "Score"
      Score = max(xgbcv$evaluation_log$test_auc_mean),

      # Get number of trees for the best performing model
      nrounds = xgbcv$best_iteration
    )

    return(lst)
  }

# create boundaries for the parameters to optimize
bounds <- list(
  eta = c(0.001, 0.4),
  max_depth = c(2L, 20L),
  min_child_weight = c(1, 50),
  subsample = c(0.1, 1),
  lambda = c(1, 10),
  alpha = c(1, 10)
)

set.seed(1000)

#perform the optimization
bayes_out <-
  bayesOpt(
    FUN = obj_func,
    bounds = bounds,
    initPoints = length(bounds) + 2,
    iters.n = 8,
    iters.k = 4,
    verbose = 2
  )

data.frame(getBestPars(bayes_out))
# look at the best values and reset the bounds to tune further

bounds <- list(
  eta = c(0.1, 0.2),
  max_depth = c(15L, 25L),
  min_child_weight = c(1, 3),
  subsample = c(0.5, 0.9),
  lambda = c(5, 7),
  alpha = c(1, 3)
)

bayes_out <-
  bayesOpt(
    FUN = obj_func,
    bounds = bounds,
    initPoints = length(bounds) + 2,
    iters.n = 8,
    iters.k = 4,
    verbose = 2
  )

#record the best parameters
pars = getBestPars(bayes_out)
opt_params <- append(
  list(
    booster = "gbtree",
    objective = "multi:softmax",
    eval_metric = "auc",
    num_class = 3
  ),
  pars
)

# extract model
xgbcv <- xgb.cv(
  params = opt_params,
  data = dtrain,
  nround = 500,
  nfold = 5,
  stratified = T,
  prediction = TRUE,
  early_stopping_rounds = 10,
  maximize = T
)

# Get optimal number of rounds
nrounds = xgbcv$best_iteration

# Fit a xgb model
mdl <- xgboost(
  data = dtrain,
  params = opt_params,
  maximize = T,
  early_stopping_rounds = 10,
  nrounds = nrounds
)

#save the model for future use
saveRDS(mdl, 'model.RDS')

#create a list of testing samples using tru and train
test = tru$`RNASeq SLID`[!(tru$`RNASeq SLID` %in% train$`RNASeq SLID`)]

#filter x.vars to only include the testing data set
test = x.vars[test, ]

#remove the status for evaluation with the new ordinal forest model
test = test[, colnames(test) != 'Status']

#predict the new values
pred = predict(m, newdata = test, type = 'class')

#combine the prediction with the true values for the testing data
final <-
  cbind(as.character(pred$ypred),
        tru[match(rownames(test), tru$`RNASeq SLID`), 'group'])
colnames(final) = c('Predicted', 'Actual')
```

```r
final = as.data.frame(final)

#make sure the factor levels are in the desired order
final$Predicted = factor(final$Predicted,
                         levels = c('high', 'medium', 'low'))
final$Actual = factor(final$Actual,
                      levels = c('high', 'medium', 'low'))

#evaluate with a confusion matrix from caret
confusionMatrix(final$Predicted, final$Actual)
```

All 6 models were saved and ultimately model 6 was chosen because of its performance in the testing data.

Part 4: Build RIGATONI package functions

We have all the elements needed to put together the RIGATONI r package. Now we must put them together to build functions that will be useful to our users.

4.1: Download STRING database connections

First we download the string protein-protein interaction database (https://string-db.org/) and change the gene names

```r
#download the file 9606.protein.actions.v10.5.txt from STRING.
#load it into R
stringdb <-
  read.table('9606.protein.actions.v10.5.txt',
             sep = '\t',
             header = T)

#create new mart using the ensembl genes
mart <- useDataset("hsapiens_gene_ensembl", useMart("ensembl"))

#get the list of genes (not isoforms) from the stringdb dataframe
genes <- gsub(".*\\.", "", stringdb$item_id_a)

#get the list of hgnc symbols from this list of genes
G_list <-
  getBM(
    filters = "ensembl_peptide_id",
    attributes = c("ensembl_peptide_id", "hgnc_symbol"),
    values = genes,
    mart = mart
  )

#remove duplicated values
G_list <- G_list[!(duplicated(G_list$ensembl_peptide_id)),]

#remove isoform information from both gene columns of stringdb
stringdb$item_id_a <- gsub(".*\\.", "", stringdb$item_id_a)
stringdb$item_id_b <- gsub(".*\\.", "", stringdb$item_id_b)
```

```r
#filter stringdb to only include interactions interesting to you
stringdb = stringdb[stringdb$mode %in% c('inhibition',
                                         'activation',
                                         'expression'),]

#replace the ensembl_peptide_id's with the hgnc symbols
for (x in 1:nrow(G_list)) {
  ids = which(stringdb$item_id_a == G_list[x, 1])
  stringdb$item_id_a = replace(stringdb$item_id_a, ids, G_list[x, 2])
  ids = which(stringdb$item_id_b == G_list[x, 1])
  stringdb$item_id_b = replace(stringdb$item_id_b, ids, G_list[x, 2])
}

#save the final table
write.table(stringdb, '9606.protein.actions.v10.5.txt')
```

After this, we need a way to filter the STRING output for a particular gene.

```r
makeGeneList <- function(gene, string = sdb) {

  #sbd is the string database file
  #gene is the name of the gene of interest
  if (!(gene %in% c(string$item_id_a, string$item_id_b)))
    stop(
      'Sadly, we do not have enough protein connections in STRING to analyze
      you gene-of-interest using runRIGATONI. \nPlease build your own gene
      list using prior knowledge or literature review.'
    )

  #filter sdb to only include rows which include the gene of interest
  sdb_goi <- string[string$item_id_a == gene |
                      string$item_id_b == gene,]

  #remove rows where the acting gene (item_id_a) is absent
  sdb_goi = sdb_goi[!(is.na(sdb_goi$item_id_a)),]

  # create the upstream database by searching for cases where the acting gene
  # is gene a, the gene being acted on is the gene of interest,
  # and the mode is expression.
  upstream = sdb_goi[sdb_goi$item_id_b == gene &
                       sdb_goi$a_is_acting == 't' &
                       sdb_goi$mode == 'expression',]

  #create one version of downstream where the gene of interest is acting
  downstream_1 = sdb_goi[sdb_goi$item_id_a == gene &
                           sdb_goi$a_is_acting == 't',]

  # create another downstream where the gene of interest is not necessarily acting,
  # but expression is not being effected
  downstream_2 = sdb_goi[sdb_goi$item_id_b == gene &
                           sdb_goi$a_is_acting == 't' &
                           sdb_goi$mode != 'expression',]
  #combine the two downstream gene lists
  downstream = c(downstream_2$item_id_a, downstream_1$item_id_b)
  downstream = downstream[downstream != '']

  #extract the gene list from the upstream data
  upstream = upstream$item_id_a
  upstream = upstream[upstream != '']

  #add both the upstream and downstream lists to a master list l
  l = list(upstream, downstream)

  #name both entries
  names(l) = c('upstream', 'downstream')

  #return the master list
  return(l)

}
```

4.2: Create the regression models themselves

After this, we need to create the regression models for the gene of interest using the control data.

```
# first use a parent function to get each regression from the
# upstream and downstream gene lists
getRegression <- function(gene_list_ppi, gene, ControlRNA) {

  #check that the length of the upstream gene list is greater than 0
  if (length(gene_list_ppi$upstream) > 0) {

    #if it is greater than 0, build the regression model for the upstream list
    RegressionUpstream <-
      buildRegression(ControlRNA, gene, gene_list_ppi$upstream)

  } else {

    #if not, call the regression model "skip"
    RegressionUpstream = 'Skip'

  }

  #check that the length of the downstream gene list is greater than 0

  if (length(gene_list_ppi$downstream) > 0) {

    #if it is greater than 0, build the regression model for the downstream list

    RegressionDownstream <-
      buildRegression(ControlRNA, gene, gene_list_ppi$downstream)

  } else {

    #if not, call the regression model "skip"
    RegressionDownstream = 'Skip'

  }

  #store each regression model in a master list
  l = list(RegressionUpstream, RegressionDownstream)

  #name each model
  names(l) = c('RegressionUpstream', 'RegressionDownstream')

  #return the list
  return(l)

}

#in this function, we build the regression models themselves
buildRegression <- function(ControlRNA, gene, gene_list) {

  #create a vector of the counts of the gene of interest called cd
  cd = t(ControlRNA[rownames(ControlRNA) == gene,])

  #name the entries after the colnames of the control RNA
  rownames(cd) = colnames(ControlRNA)

  #turn the vector into a dataframe
  cd = as.data.frame(cd)

  #create the x.vars as a dataframe with genes as columns and sample names as rows
  #the genes in this dataframe should only be in the gene_list provided
  x.vars = as.data.frame(t(ControlRNA[rownames(ControlRNA) %in% gene_list,]))

  #combine the x.vars dataframe with the gene of interest counts
  x.vars = cbind(x.vars, cd[, 1])

  #rename the gene of interest column GOI
  colnames(x.vars)[ncol(x.vars)] = 'GOI'

  #replace any NA entries with 0
  x.vars[is.na(x.vars)] = 0

  # create the model using the GOI column as a response and all
  # other columns as predictors using a poisson distribution
  model = stats::glm(GOI ~ ., data = x.vars, family = stats::poisson())

  #return the model
  return(model)

}
```

4.3: Predict the function of each mutant sample

After creating the models, we need to use the models to predict the functional status of each mutant sample

```
# this function uses the regression models created previously (Regression)
# and a RNA from the mutant samples (MasterRNA) to determine the the gene of
# interest (gene) count is outside the 95% prediction interval for
# each regression model
mutantRegression <- function(Regression, MasterRNA, gene) {

    if (Regression$RegressionUpstream != 'Skip') {
    #generate the gene names in the upstream regression
    names_up = names(Regression$RegressionUpstream$coefficients)

  } else {

    #if the upstream regression was skipped, make the names_up vector empty
    names_up = c()

  }
  if (Regression$RegressionDownstream != 'Skip') {
    #generate the gene names in the downstream regression
    names_down = names(Regression$RegressionDownstream$coefficients)

  } else {

    #if the downstream regression was skipped, make the names_up vector empty
    names_down = c()

  }

  #store the genes in upstream and downstream together in gene_list
  gene_list <- c(names_up, names_down)
  rownames(MasterRNA) = gsub("\\-", "\\.", rownames(MasterRNA))

  #store the regressions respectively
  upstream = Regression$RegressionUpstream
  downstream = Regression$RegressionDownstream

  #create a dataframe with the counts of the gene of interest (cd)
  cd = t(MasterRNA[rownames(MasterRNA) == gene,])

  #save the sample names as rownames of the GOI dataframe (cd)
  rownames(cd) = colnames(MasterRNA)
  cd = as.data.frame(cd)

  #create the x.vars as a dataframe with genes as columns and sample names as rows
  #the genes in this dataframe should only be in the gene_list provided
  x.vars = as.data.frame(t(MasterRNA[rownames(MasterRNA) %in% gene_list,]))

  #rename the gene of interest column GOI
  x.vars$GOI = cd[, 1]

  #create prediction intervals for each sample, for each regression model
  #alpha = .1 means you are creating a 95% prediction interval
  #nsims is the number of simulations of prediction to complete
  if (upstream != 'Skip') {
```

```
  fitted_up <-
    suppressWarnings(ciTools::add_pi(x.vars, upstream, alpha = 0.1, nsims = 20000))

} else {

  #if no upstream model is present, skip it
  fitted_up <- 'Skip'

}
if (downstream != 'Skip') {

  fitted_down <-
    suppressWarnings(ciTools::add_pi(x.vars, downstream, alpha = 0.1, nsims = 20000))

} else {

  #if no downstream model is present, skip it
  fitted_down <- 'Skip'

}

#put the fittered dataframes in to a list called l
l = list(fitted_up, fitted_down)

#for each dataframe in l
l = lapply(l, function(f) {

  if (f != 'Skip') {

    #as long as the entry of l is not "skip"
    #select only the columns listed below
    f = f[, colnames(f) %in% c('GOI', 'LPB0.05', 'UPB0.95')]

    #initialize a new vector called anno
    anno = c()

    for (x in 1:nrow(f)) {

      #for each sample
      #if the true GOI count is outside the 95% prediction interval mark it F
      if (f$GOI[x] < f$LPB0.05[x] |
          f$GOI[x] > f$UPB0.95[x]) {

        anno = c(anno, F)

      } else {

        anno = c(anno, T)

      }
    }

    #add a column to f called Annotation with the entries of anno
```

```
      f$Annotation = anno

      return(f)

    } else {

      #if the model was empty, skip it
      return('Skip')

    }
  })

  #rename the entries of l
  names(l) = c('upstream', 'downstream')

  #return l
  return(l)
}

#This function determines the function of each sample
getMutantFunction <- function(Regression, MasterRNA, gene) {

  # first the function using mutantRegression function (above) to
  # determine which samples are outside the 95% prediction interval
  Mutant_Regression <-
    suppressWarnings(mutantRegression(Regression, MasterRNA, gene))

  #remove the entries of the list that we should "skip"
  Mutant_Regression_1 = Mutant_Regression[Mutant_Regression != 'Skip']

  if (length(Mutant_Regression_1) > 1) {

    # if both regression models are working, combine them into a
    # master data frame called Anno
    Anno <- do.call(cbind, Mutant_Regression)
    Anno = as.data.frame(Anno)

  } else {

    #if at one regression model is skipped, rename the remaining model "Anno"
    Anno = Mutant_Regression_1[[1]]

  }
  #initialize empty vector fun
  fun <- c()

  if (Mutant_Regression$downstream != 'Skip') {

    #if the downstream regression was able to be created continue below
    for (x in 1:nrow(Anno)) {

      #for each sample
      #create new vector with the GOI counts
      downstream.GOI = ifelse((T %in% grepl('downstream', colnames(Anno))),
                              Anno$downstream.GOI[x],
                              Anno$GOI[x])

      #create new vector with lower prediction interval bound
      downstream.LPBO.05 = ifelse((T %in% grepl('downstream', colnames(Anno))),
                                  Anno$downstream.LPBO.05[x],
                                  Anno$LPBO.05[x])

      #create new vector with upper prediction interval bound
      downstream.UPBO.95 = ifelse((T %in% grepl('downstream', colnames(Anno))),
                                  Anno$downstream.UPBO.95[x],
                                  Anno$UPBO.95[x])

      if (downstream.GOI < downstream.LPBO.05) {

        #if the true value is less than the lower bound, this is a LOF sample
        fun = c(fun, 'LOF')

      } else if (downstream.GOI > downstream.UPBO.95) {

        #if the true value is greater than the upper bound, this is a GOF sample
        fun = c(fun, 'GOF')

      } else {

        # if the downstream predictions are within the boundaries,
        # keep checking with the upstream samples
        fun = c(fun, 'keepchecking')

      }
    }

    #remove the solved samples, call them solved
    AnnoSolved <- Anno[which(fun != 'keepchecking'),]

    #create new column with the function data
    AnnoSolved$Function = fun[which(fun != 'keepchecking')]

    #put the samples that need to be rechecked into their own dataframe
    AnnoKeep <- Anno[which(fun == 'keepchecking'),]

  } else {

    #if there are no downstream values, annosovled should be empty
    AnnoSolved = NULL

    #annokeep should be the entirety of anno
    AnnoKeep = Anno

  }
  if (Mutant_Regression$upstream != 'Skip') {
    #if the upstream regression is present
    #initialize the empty fun vector
    fun = c()

    #as long as the downstream didn't solve everything
    if (nrow(AnnoKeep > 0)) {

      for (x in 1:nrow(AnnoKeep)) {

        #for each sample
        #create new vector with the GOI counts
        upstream.GOI = ifelse((T %in% grepl('upstream', colnames(Anno))),
                              AnnoKeep$upstream.GOI[x],
                              AnnoKeep$GOI[x])

        #create new vector with lower prediction interval bound
        upstream.LPB0.05 = ifelse((T %in% grepl('upstream', colnames(Anno))),
                                  AnnoKeep$upstream.LPB0.05[x],
                                  AnnoKeep$LPB0.05[x])

        #create new vector with upper prediction interval bound
        upstream.UPB0.95 = ifelse((T %in% grepl('upstream', colnames(Anno))),
                                  AnnoKeep$upstream.UPB0.95[x],
                                  AnnoKeep$UPB0.95[x])

        if (upstream.GOI < upstream.LPB0.05) {

          #If the true value is less than lowerbound, this is a LOF sample
          fun = c(fun, 'LOF')

        } else if (upstream.GOI > upstream.UPB0.95) {

          #if the true value is greater than the upperbound, this is a GOF sample
          fun = c(fun, 'GOF')

        } else {

          #if we cannot decide, call it skipq
          fun = c(fun, "Skipq")

        }
      }
    }

    #replace skipq with "unknown
    fun[fun == 'Skipq'] = 'Unknown'

    #create new column called Function with the vector
    AnnoKeep$Function = fun

  }

  if (!('Function' %in% colnames(AnnoKeep))) {
    # if we had no upstream regression, initialize columnn Function with
    # all entries "unknown"
    AnnoKeep$Function = rep('Unknown', nrow(AnnoKeep))

  }

  #combine the solved and kept dataframes into Mutant_Function
  Mutant_Function <- rbind(AnnoSolved, AnnoKeep)

  #remove extraneous columns
  Mutant_Function <-
    as.data.frame(cbind(rownames(Mutant_Function), Mutant_Function$Function))

  #return mutant function
  return(Mutant_Function)

}
```

4.4: Predict the immune phenotype of each sample

```
# predict the immune phenotype using the genes selected by Elastic Net (lasso_best)
# using the model created by Ordinal forect (m) and the RNA seq data of the
# mutant samples (MasterRNA)
getImmuneProb <- function(MasterRNA, model = m) {

  #filter MasterRNA so that the only genes are those that are in the model
  MasterRNA = MasterRNA[rownames(MasterRNA) %in% m$feature_names,]

  #make sure the order of the genes match
  MasterRNA = MasterRNA[match(m$feature_names, rownames(MasterRNA)),]

  #make the prediction
  pred <- predict(m, newdata = t(MasterRNA), type = 'class')

  #Add sample names to the prediction
  pred = cbind(colnames(MasterRNA), pred)

  #create data frame
  pred = as.data.frame(pred)

  #add column names
  colnames(pred) = c('Names', 'Prob')

  #annotate with meaningful naems
  pred$Prob = ifelse(pred$Prob == 0, 'Low', ifelse(pred$Prob == 2, 'High', 'Medium'))

  #return data
  return(pred)

}
```

4.5: Evaluate mutants for users

```
#evaluate the mutant samples provided by the user
evaluateMutants <- function(Function, ImmuneProb, ControlRNA) {

  # get the immune phenotypes of the control samples
  con = suppressMessages(getImmuneProb(ControlRNA, model = m))

  # get the number of GOF and LOF mutant samples
  nGOF = nrow(Function[Function[, 2] == 'GOF',])
  nLOF = nrow(Function[Function[, 2] == 'LOF',])

  # get the number of High and Low mutant samples
  nHigh = nrow(ImmuneProb[ImmuneProb[, 2] == 'High',])
  nLow = nrow(ImmuneProb[ImmuneProb[, 2] == 'Low',])

  # Get the proportion of High and Low control samples
  conPropH = nrow(con[con[, 2] == 'High']) / nrow(con)
  conPropL = nrow(con[con[, 2] == 'Low']) / nrow(con)

  # determine if there are more GOFs than 50%
  pf = prop.test(nGOF, sum(nGOF, nLOF), .5)

  # record this result in an intuitive way
  Fun = ifelse(pf$p.value < .05, ifelse(nGOF > nLOF, 'GOF', 'LOF'), 'Unknown')

  # determine if there is higher proportion of high or low samples
  # in the mutant group than in controls
  pih = prop.test(nHigh, nrow(out), conPropH, 'greater')
  pil = prop.test(nLow, nrow(out), conPropL, 'greater')

  # record this result in an intuitive way
  Imm = ifelse(pih$p.value < .05,
               'High',
               ifelse(pil$p.value < .05, 'Low', 'Unknown'))

  # record the p values
  pi = ifelse(Imm == 'High',
              pih$p.value,
              ifelse(Imm == 'Low', pil$p.value, min(pil$p.value, pih$p.value)))

  #store and return these values
  out = list(Fun, pf, Imm, pi)
  names(out) = c('Function', 'p.val.func', 'ImmunePhenotype', 'p.val.immune')
  return(out)
}
```

4.6. Run RIGATonI all together

```
runRIGATONI <- function(gene,
                        ControlRNA,
                        MasterRNA,
                        model = m,
                        string = sdb,
                        geneList = NULL) {

  # catch any issues with the function prior
  if (any(class(ControlRNA) != 'data.frame',
          class(MasterRNA) != 'data.frame'))
    stop('RNA files should be data.frame objects')

  if (any(!(rownames(ControlRNA) == rownames(MasterRNA))))
    stop('Gene names in ControlRNA are not the same as MasterRNA')

  if (!(gene %in% rownames(ControlRNA)))
    stop('Gene of interest TPM data is not included in RNA provided')

  if (!(m$feature_names %in% rownames(MasterRNA)))
    stop(
      'The genes required for immune phenotype calculation are not
      all present in the data provided. \nPlease check that all your gene names
      are uppercase and the rownames of your RNA TPM data.'
    )

  message(paste0('Starting gene: ', gene))

  if (is.null(geneList) == T) {

    # if there is no gene list provided by the suer, make one
    message('Making Initial Gene List')
    gene_list_ppi = makeGeneList(gene, string = sdb)

    # filter the gene list to include only genes for which there is
    # transcript information in the control RNA
    message('Filtering Gene List')
    gene_list_ppi = lapply(gene_list_ppi, function(x) {

      out = x[x %in% rownames(ControlRNA)]

      return(out)

    })

    # if there are no connections in string, tell the user
    if (length(unlist(gene_list_ppi)) > 0)
      stop(
        'Sadly, we do not have enough protein connections in STRING to analyze
        your gene-of-interest using runRIGATONI. \nPlease build your own gene
        list using prior knowledge or literature review.'
      )

  } else {

    # if the user provided their own gene list, store it
    gene_list_ppi = geneList
    message('Filtering Gene List')

    # filter the gene list to include only genes for which there is
    # transcript information in the control RNA
    gene_list_ppi = lapply(gene_list_ppi, function(x) {

      out = x[x %in% rownames(ControlRNA)]

      return(out)
    })

    # if there is nothing left after filtering, tell the user to try
    # again with default settings
    if (length(unlist(gene_list_ppi)) > 0)
      stop(
        'Your gene list is either empty or does not contain genes in your RNA
        TPM data. \nPlease re-run RIGATONI with default settings and allow us
        to make the geneList using STRING.'
      )

  }
  # build control regression model
  message('Building regression model')
  Regression <-
    suppressWarnings(getRegression(gene_list_ppi, gene, ControlRNA))

  # predict mutant function
  message('Predicting mutant function')
  Mutant_Function <-
    suppressWarnings(getMutantFunction(Regression, MasterRNA, gene))

  # predict immune phenotypes
  message('Predicting immune phenotype')
  ImmuneProb <- suppressMessages(getImmuneProb(MasterRNA, model = m))
  con = suppressMessages(getImmuneProb(MasterRNA, model = m))

  # put together final output for the user
  message('Getting final output')
  out = cbind(colnames(MasterRNA), Mutant_Function[, 2], ImmuneProb[, 2])
  colnames(out) = c('SampleID', 'Function', 'ImmunePhenotype')
  out = as.data.frame(out)

  # calculate the mutant function and phenotype
  message('Calculating Alteration Function and Phenotype')
  muts = evaluateMutants(Mutant_Function, ImmuneProb, ControlRNA)

  # inform the user of the results in an intuitive way
  message(cat(

    paste0(
      'Function of Variant: ',
      muts$Function,
      ", p = ",
      muts$p.val.func,
      "\n",
      "Immune Phenotype of Variant: ",
      muts$ImmunePhenotype,
      ", p = ",
      muts$p.val.immune
    )

  ))

  #return results to the user
  message('Done!')
  return(out)
}
```

Part 5: Performing RIGATonI validations (Fig 2 and 3)

5.1: Validation of Immunity Module with IHC and Flow (Fig 2)

All data shown here is from the paper "https://www.nature.com/articles/s41598-022-12610-w#Abs1" First we downloaded the raw fastqs from dbGaP (the accession number is provided under "Data Availability" in the paper). Next we ran Salmon to extract RNA TPM from the fastqs; code shown below.

```
#!/bin/sh
#SBATCH --account=PAS0854
#SBATCH --time=10:00:00
#SBATCH --nodes=1
#SBATCH --ntasks=10

echo "<STARTING_PIPELINE>: $(date)"  &&  \

mkdir -p Salmon  &&  \
cd Salmon  &&  \


# Copy input files to $TMPDIR
echo "------------------------------------------------------------"  &&  \
echo -ne "$(date): Starting command 'cp  Sample 1 Run'\n"  &&  \
cp  sample_*.fastq $TMPDIR  &&  \
echo -ne "$(date): Ending command 'cp  Sample 1 Run'\n"  &&  \
echo "------------------------------------------------------------"  &&  \

cd $TMPDIR  &&  \
echo "Working on $(pwd) with input files:"  &&  \
echo "$(ls)"  &&  \

echo "------------------------------------------------------------"  &&  \
echo -ne "$(date): Starting command 'salmon quant salmon_index'\n"  &&  \
Software/salmon/salmon-latest_linux_x86_64/bin/salmon quant  \
-i Reference_Data/salmon_reference_data_20190517/salmon_index -l A  \
-1 sample_1.fastq  \
-2 sample_2.fastq -p 10 --validateMappings -o salmon_results  &&  \
echo -ne "$(date): Ending command 'salmon quant salmon_index'\n"  &&  \
echo "------------------------------------------------------------"  &&  \

echo "------------------------------------------------------------"  &&  \
echo -ne "$(date): Starting command 'mv quant.sf .'\n"  &&  \
mv salmon_results/quant.sf .  &&  \
echo -ne "$(date): Ending command 'mv quant.sf .'\n"  &&  \
echo "------------------------------------------------------------"  &&  \

rm -rf salmon_results  &&  \
echo "------------------------------------------------------------"  &&  \
echo -ne "$(date): Starting command 'bedtools intersect'\n"  &&  \
Software/bedtools-2.17.0/bin/bedtools intersect \
-a Reference_Data/bedfiles/dna/exome_bedfiles/sorted_merged.bed \
-b Reference_Data/refFlat_hg19.sorted.bed -wa -wb \
> targetbed_refflat_intx.bed  &&  \
echo -ne "$(date): Ending command 'bedtools intersect'\n"  &&  \
echo "------------------------------------------------------------"  &&  \

echo "------------------------------------------------------------"  &&  \
echo -ne "$(date): Starting command 'python annotate_salmon.py'\n"  &&  \
python annotate_salmon.py --in_file quant.sf \
--gtf Reference_Data/cuffcmp.combined.converted.gtf \
--bed_refflat_intx targetbed_refflat_intx.bed  &&  \
echo -ne "$(date): Ending command 'python annotate_salmon.py'\n"  &&  \
echo "------------------------------------------------------------"  &&  \
```

Once salmon has run, we use a python script (annotate_salmon.py) to transform RefSeq ID to gene names shown below

```
"""
The purpose of this script is to get the gene names for the Salmon output,
since Salmon only provides RefSeq IDs. It will mark whether the gene is in
the BED file.
"""

import sys
import string
import os
import getopt
import argparse
import glob

##########################################################################
## Parse command line arguments
##########################################################################

class DefaultHelpParser(argparse.ArgumentParser):
  def error(self, message):
    sys.stderr.write('ERROR: %s\n' % message)
    self.print_help()
    sys.exit(2)
```

```
parser = DefaultHelpParser(formatter_class=argparse.RawTextHelpFormatter)
parser.add_argument( '--in_file', required = True, help = 'Input file that was '+\
                     'produced from Salmon.')
parser.add_argument('--gtf', required = True, help = 'GTF file for transcript a'+\
                    'nnotation.')
parser.add_argument('--bed_refflat_intx', required = True, help = 'BEDTools '+\
                    'intersection of the target-BED with the refFlat BED.')
args = parser.parse_args( sys.argv[1:])

##########################################################################
## Function definitions
##########################################################################

"""
The purpose of this function is to get a list of genes within the input BED
region using the intersection with the input BED and the refFlat BED.
"""
def get_in_target_genes(bed_refflat_intx):
  bed_genes_set = set()

  for line in open(bed_refflat_intx):
    line1 = line.strip().split('\t')
    gene = line1[-1].split('_')[0]
    bed_genes_set.add(gene)

  return bed_genes_set


"""
The purpose of this function is to collect the RefSeq IDs and match them
to the gene name.
"""
def get_refseq_ids(gtf):
  refseq_dict = {}

  for line in open(gtf):
    line1 = line.strip().split('\t')
    annots = line1[8].split(';')

    for entry in annots:
      if entry == '': continue
        entry1 = entry.strip()

      if entry1.startswith('gene_name'):
        entry2 = entry1.split(' ')
        gene_name = entry2[1].replace('"', '')

      elif entry1.startswith('transcript_id'):
        entry2 = entry1.split(' ')
        transcript_id = entry2[1].replace('"', '')

    refseq_dict[transcript_id] = gene_name
  return refseq_dict


"""
The purpose of this function is to annotate the Salmon output file with gene
names.
"""
def annotate(in_file, refseq_dict, bed_genes_set):
  fp = open("Salmon_final.txt", 'w')
  header = ["Gene", "RefSeq ID", "TPM", "Gene In Target"]
  fp.write(string.join(header, '\t') + '\n')

  all_values = []

  for c,line in enumerate(open(in_file)):
    if c==0: continue
    line1 = line.strip().split('\t')

    tpm = float(line1[3])
    tpm = float('%4.2f' % tpm)
    refseq_id = line1[0]

    try:
      gene = refseq_dict[refseq_id]
    except KeyError: continue

    info = [gene, refseq_id, tpm, gene in bed_genes_set]
    all_values.append(info)

  all_values.sort(key=lambda x:x[2])
  all_values.reverse()

  for info in all_values:
    info1 = map(lambda x:str(x), info)
    fp.write(string.join(info1, '\t') + '\n' )

  fp.close()


##########################################################################
## Main program
##########################################################################

if __name__ == '__main__':
  in_file = os.path.abspath(args.in_file)
  gtf = os.path.abspath(args.gtf)
  bed_refflat_intx = os.path.abspath(args.bed_refflat_intx)

  bed_genes_set = get_in_target_genes(bed_refflat_intx)
  refseq_dict = get_refseq_ids(gtf)
  annotate(in_file, refseq_dict, bed_genes_set)
```

After downloading and performing salmon, I then combined and analyzed the IHC, flow, and RNAseq results.

Processed data was downlaoded from the supplemental data at this link https://static-content.springer.com/esm/art%3A10.1038%2Fs41598-022-12610-w/MediaObjects/41598_2022_12610_MOESM2_ESM.xlsx.

```
# read in the data
gas = lapply(2:15, function(x) {
  readxl::read_xlsx('gastric_IHC.xlsx', sheet = x)
})

# name each element of the list
names(gas) <- c(
  'Patients characteristics (BKT cohort)',
  'Transcriptome-based immune cell quantification',
  'IHC-based immune cell quantification',
  'FCM-based immune cell quantification',
  'The ratio of indicated cell densities at the Core
  (CT)-to-invasive margin (IM) of the tumor',
  'TIDE dysfunction and exclusion scores',
  'The percentage of cytokineproducing cells',
  'Individual patient´s Ki-67 related score measured by IHC',
  'The number of damaged cells in 12 areas of 0.25 mm2',
  'Individual patients data (TCGA cohort)',
  'Comparison between Immunogram classification and transcriptome-based
  TME classification',
  'Gene sets used in this study',
  'Correspondence between the subtypes of bulk RNA-seq immune cells
  estimation and the values measured by IHC',
  'Correspondence between the subtypes of bulk RNA-seq immune cells
  estimation and the values measured by FCM'
)

# save for future use
saveRDS(gas, 'gastric_IHC.RDS')

# gather the processed TPM data
d = list.dirs('<path to TPM data>')
d = d[grepl('Salmon', d)]
d = d[d != '<scratch directory>']

# read in each file
t = lapply(d, function(x) {

  tryCatch({
    # list the files in the folder
    l = list.files(x)

    # select only the final output
    l = l[grepl('_final', l)]

    #read it in and summarize
    t = read.table(paste0(x, '/', l[1]), header = T, sep = '\t')
    t = aggregate(t[, 3], by = list(t[, 1]), sum)
    colnames(t)[2] = strsplit(l, '_')[[1]][2]

    #return
    return(t)

  }, error = function(e) {

    #if the file is unreadable return nothing
    return(NA)
  })
})

# remove file which are not needed
t = t[1:57]

# make a key of the gene names
key = t[[1]]$Group.1

# ensure each file has the same order of genes
t2 = lapply(1:length(t), function(x) {

  #using the first file as a key
  if (x == 1) {

    # just return the first one
    return(t[[x]])

  } else {

    # match the genes to the first file
    x = t[[x]]
    nam = colnames(x)[2]
    x = x[match(key, x$Group.1),]
    x = as.data.frame(x[, 2])
    colnames(x) = nam

    return(x)
  }
})

# combine together all the files
t2 = do.call(cbind, t2)

# set the rownames to the gene names located in the first column
rownames(t2) = t2[, 1]
t2 = t2[,-1]
rna = t2

rm(t2)
rm(t)
rm(d)
rm(key)

# there were many sample IDs in this data set which did not match across
# groups. The next section is primarily for organizing these sample IDs across
# the different data tables available
```

```r
# read in the sample IDs
ids = readLines('GastricRNA.txt')

# keep only RNA which have accompanying IHC and flow
rna = rna[, colnames(rna) %in% ids]

# read in the run table
ids = read.table('SraRunTable.txt', sep = ',', header = T)

# filter the run table to inclue only runs in the RNA
ids = ids[ids$Run %in% colnames(rna),]

# match the run table order to the RNA order
ids = ids[match(colnames(rna), ids$Run),]

# extract clinical data from the excel files
clin = gas$`Patients characteristics (BKT cohort)`

#remove rows with no study ID
clin = clin[!(is.na(clin$`Study ID`)),]

# get the new study IDs
studyID = readxl::read_xlsx('BKT_STAD.xlsx')

# add in the study ID to the clinical data
clin$STAD = unlist(lapply(clin$`Study ID`, function(x) {

  id = which(studyID$`patient ID` == x)

  return(studyID$`file ID`[id])
}))

# add in run information to the clinical data
clin$run = unlist(lapply(clin$STAD, function(x) {

  id = which(gsub("\\..*", '', ids$Sample_Name) == x)

  return(ids$Run[id])
}))

# make the RNA columns match the order of the clinical data
rna = rna[, match(clin$run, colnames(rna))]

# change the column names of the RNA to match the study ID
colnames(rna) = clin$`Study ID`

# add the TPM to the gastric IHC object
gas$TPM = rna

# save for future use
saveRDS(gas, 'gastric_IHC.RDS')

# get the gastric data RIGATonI scores
```

```r
riga = getImmuneProb(gas$TPM, m)
# add that output to the clincal data
checking = !(is.na(gas$`Patients characteristics (BKT cohort)`$`Study ID`))
gas$`Patients characteristics (BKT cohort)` =
  gas$`Patients characteristics (BKT cohort)`[checking,]
gas$`Patients characteristics (BKT cohort)`$RIGATONI_score = riga$Prob

# save for future use
saveRDS(gas, 'gastric_IHC.RDS')

# extract the IHC information
ihc = gas$`IHC-based immune cell quantification`

# extract the measure names
measure = colnames(ihc)

# fix the column names
colnames(ihc) = ihc[1,]
ihc = ihc[-1,]

# make IHC a data frame
ihc = as.data.frame(t(ihc))

# fix the new column names
colnames(ihc) = ihc[1,]
ihc = ihc[-1,]

# get measures of interest
measure = measure[2:length(measure)]

# make sure measures are readable
measure = gsub("\\..*", "", measure)

# add measures as a new column
ihc$measure = measure

# split IHC based on the measures
ihc = split(ihc, ihc$measure)

# now we will make the data more readable
ihc = lapply(1:length(ihc), function(x) {
  # for each measure
  # record the measure
  mes = names(ihc)[x]
  # make new data frame
  x = ihc[[x]][,-ncol(ihc[[x]])]
  nams = colnames(x)
  # transpose and convert to numeric
  x = as.data.frame(t(x))
  x = apply(x, 2, as.numeric)
  x = as.data.frame(x)
  colnames(x) = gsub("\\..*", "", colnames(x))
  # change the name of one column
  if ('Nkp46' %in% colnames(x)) {
    id = which(colnames(x) == 'Nkp46')
    colnames(x)[id] = 'NKp46'
  }
  rownames(x) = nams
  # add the rigatoni status
  x$RIGATONI_status = gas$`Patients characteristics (BKT cohort)`$RIGATONI_score
  # add the measure as a varaible
  x$measure = mes
  return(x)
})

# extract counts
count = ihc[[2]]

# melt the counts
countm = reshape::melt(count, id.vars = 'RIGATONI_status')

# aggregate based on measure and RIGATonI status
countm = aggregate(value ~ variable + RIGATONI_status,
                   data = countm,
                   FUN = mean)

# fix order of values
countm$RIGATONI_status = factor(countm$RIGATONI_status, levels = c('Low', 'High'))

# Figure 2C
g = ggplot(countm, aes(x = RIGATONI_status, y = value, fill = variable)) +
  geom_bar(stat = 'identity') +
  scale_fill_manual(values = grDevices::colors()[grep('gr(a|e)y',
                                                       grDevices::colors(),
                                                       invert = T)] %>%
                      sample(10, replace = FALSE)) +
  theme_classic()

saveRDS(g, '<figure 2c>')

# remake the melted data
countm = reshape::melt(count, id.vars = 'RIGATONI_status')

# Figure 2D
cd8 = countm[countm$variable == 'CD8',]

cd8$RIGATONI_status = factor(cd8$RIGATONI_status, levels = c('Low', 'High'))

g = ggplot(cd8, aes(x = RIGATONI_status, y = value, fill = RIGATONI_status)) +
  geom_boxplot() +
  theme_classic() +
  stat_compare_means(
    label = 'p.signif',
    label.x = 1.5,
    size = 10,
    label.y = 4e+05
  )

saveRDS(g, '<figure 2d>')

# Figure 2F
cd3 = countm[countm$variable == 'CD3',]

cd3$RIGATONI_status = factor(cd3$RIGATONI_status, levels = c('Low', 'High'))

g = ggplot(cd3, aes(x = RIGATONI_status, y = value, fill = RIGATONI_status)) +
  geom_boxplot() +
  theme_classic() +
  stat_compare_means(
    label = 'p.signif',
    label.x = 1.5,
    size = 10,
    label.y = 4e+05
  )

saveRDS(g, '<figure 2f>')

# now we will do the flow data
# extract the flow data
flow = gas$`FCM-based immune cell quantification`

# fix the colnames so they are not the first row
colnames(flow) = flow[1,]
flow = flow[-1,]

# add in the RIGATonI scores
flow$RIGATONI = gas$`Patients characteristics (BKT cohort)`$RIGATONI_score

# melt the data
flow = reshape2::melt(flow, id.vars = 'RIGATONI')

# remove the study IDs from the variables
flow = flow[flow$variable != 'Study ID',]

# transform the numbers from characters to numerics
flow$value = as.numeric(as.character(flow$value))

# remove NA values
flow = flow[!(is.na(flow$value)),]

# fix the factor levels
flow$RIGATONI = factor(flow$RIGATONI, levels = c('Low', 'High'))

# change the levels to be more readable
levels(flow$variable) = c(
  'studyID',
  'Lymphocytes',
  'CD4+ T cells',
  'CD8+ T cells',
  'B Cells',
  'NK Cells',
  'Monocytes\nMacrophages'
)

# Figure 2E
test = flow[flow$variable == 'Lymphocytes', ]

g = ggplot(test, aes(x = RIGATONI, y = value, fill = RIGATONI)) +
  geom_boxplot() +
  stat_compare_means(
    label = 'p.signif',
    size = 5,
    label.x = 1.5,
    method.args = list(alternative = 'greater')
  ) +
  theme_classic()

saveRDS(g, '<Figure 2E>')
```

5.2: Validation of Immunity Module with Saltz Et Al output (Fig 2)

All data shown here is from the paper "https://www.sciencedirect.com/science/article/pii/S2211124718304479?via%3Dihub" First we downloaded the output from Saltz et al found at "https://ars.els-cdn.com/content/image/1-s2.0-S2211124718304479-mmc2.xlsx" Then the following code was run.

```
# read in the saltz data containing TIL, case ID, and spatial categorization
sal = read.csv('saltzdata.csv')

# convert the barcodes to UUIDs to match our master data
sal$caseIDs = barcodeToUUID(sal$ParticipantBarcode)[, 2]

# load the list of cases and phenotypes
riga = readRDS('<TCGA all scores>')

# make the order and cases match for both data tables
riga = riga[riga$caseIDs %in% sal$caseIDs,]
sal = sal[sal$caseIDs %in% riga$caseIDs,]
riga = riga[match(sal$caseIDs, riga$caseIDs),]

# add the RIGATonI scores to the saltz output
sal$riga = riga$Prob

#Figure 2G
g = ggplot(sal, aes(x = riga, y = til_percentage, fill = riga)) +
  geom_boxplot() +
  stat_compare_means(
    label = 'p.signif',
    size = 5,
    label.x = 1.5,
    method.args = list(alternative = 'greater'),
    comparisons = list(c('Low', 'Medium'),
                      c('Medium', 'High'),
                      c('Low', 'High'))
  ) +
  theme_classic()

saveRDS(g, '<Figure 2G>')

# Figure 2H
g = ggplot(sal, aes(x = riga, fill = Global_Pattern)) +
  geom_bar(position = 'fill') +
  scale_fill_manual(values = grDevices::colors()[grep('gr(a|e)y',
                                                       grDevices::colors(),
                                                       invert = T)] %>%
                     sample(6, replace = FALSE)) +
  theme_classic()

saveRDS(g, '<Figure 2H>')
```

5.3: Validation of Function Module with OncoKB (Fig 3)

First, the Function Module was run using the functions described in section 4 on OncoKB genes. Results were manually annotated using the OncoKB database. The complete results can be found in Supplementary table 3.

```
# read in the genomic alteration annotations
onco = read.csv('SupplementaryTable3.csv')

# remove any that are unknown to oncokb
onco = onco[onco$`OncoKB annotation` != 'unk',]

#mark correct or incorrect for each GA
onco$correct = ifelse(
  onco$RIGATONI_annotation  == 'GOF',
  ifelse(onco$`OncoKB annotation` == 'GOF', 'Correct', 'Incorrect'),
  ifelse(onco$`OncoKB annotation` == 'LOF', 'Correct', 'Incorrect')
)

# Figure 3B
g = ggplot(onco, aes(x = `OncoKB annotation`, fill = correct)) +
  geom_bar(position = 'fill') +
  theme_classic()

saveRDS(g, '<Figure 3B>')
```

Part 6: Analyze all of TCGA using RIGATONI

6.1: Create functions specific to the analysis of TCGA data

I created a series of functions to make analysis of TCGA data easier. They are shown below.

```r
# In makeMaster, filterMaster, addPrimarySite, addSex, and getMaster,
# we parse the lists created in Part 1 to create a Master data frame
# Each row of the dataframe represents and individual alteration.
# each row contains the name of the sample, the kind of alteration,
# the name of the alteration, the location of the alteration and primary
# site of the tumor.

makeMaster <- function(input_dir, gene) {

  #go to the input_dir and collect the mutation files
  setwd(input_dir)
  l = lapply(list.files(pattern = "Together2.RDS"), function(x)
    readRDS(x))
  names(l) = list.files(pattern = "Together2.RDS")

  #put the results from your GOI into a list called alts
  alts = list()
  nams = c()
  for (x in 1:length(l)) {

    # check to see if there are any alterations in that gene of the given type
    if (T %in% grepl(gene, names(l[[x]]))) {

      # if there are, extract them
      nam = names(l)[x]
      id = which(grepl(gene, names(l[[x]])) == T)

      if (nam == 'FusTogether2.RDS') {

        # fusions are special and have a different  rule
        # create new data frame with the fusion results results
        fusions = names(l[[x]])[id]

        # split so the genes involved are listed separately
        fusions = strsplit(fusions, "-")

        id = lapply(fusions, function(x) {

          # find the fusions in the given gene
          test = which(x == gene)

          if (length(test) > 0) {
            # return that list
            return(x)

          } else {

            # if there are none, return nothing
            return(NA)

          }
        })
        # get final list of list indexes
        id = id[!(is.na(id))]

        # concatenate together the IDs of interest
        id = unlist(lapply(id, function(i)

          paste0(i, collapse = "-")))

        if (length(id) > 0) {

          # get all the lists together
          alts = append(alts, l[[x]][id])

          # record the names of the alteration
          nams = c(nams, paste0(rep(nam, length(id)), ".", 1:length(id)))

        }
      } else {

        # get the ID that exactly matches the gene
        id = which(names(l[[x]]) == gene)

        # if there is at least 1 ID
        if (length(id) > 0) {

          # add the new element to the alts list
          alts[[length(alts) + 1]]  = l[[x]][[id]]

          #record the name
          nams = c(nams, nam)

        }
      }
    }
  }

#name the elements of alts after their file names
names(alts) = nams

#record the caseIDs of each list in alts
CaseIDs = c()
for (x in alts) {
  CaseIDs = c(CaseIDs, x$CaseID)
}

#now collect the necesary fields from each dataframe in the list
#the fields are as follows
# SV - c('POS', 'END', 'ALT')
# CNV - c("copy_number", 'start', 'end')
# SNV - c('Alt', 'Position')
# Fusions - c('gene1', 'gene2', 'br1', 'br2')
SampleAlts = c()
Types = c()
```

```
  Places = c()
  for (x in names(alts)) {

    # put together the names of the alterations according the rules described
    # for each alteration type
    if (x == 'SVTogther2.RDS') {

      place = paste0(alts[[x]]$POS, "-", alts[x]$END)
      alt = paste0(alts[[x]]$ALT)
      type = rep('SV', length(alt))

    } else if (x == "CopyTogether2.RDS") {

      place = paste0(alts[[x]]$start, "-", alts[x]$end)
      alt = paste0(alts[[x]]$copy_number)
      type = rep('CNV', length(alt))

    } else if (x == "MafsTogether2.RDS") {

      place = alts[[x]]$Position
      alt = paste0(alts[[x]]$Alt)
      type = rep('SNV', length(alt))

    } else {

      place = paste0(alts[[x]]$breakpoint1, "-", alts[[x]]$breakpoint2)
      alt = paste0(alts[[x]]$gene1, "-", alts[[x]]$gene2)
      type = rep('FUS', length(alt))

    }
    # record the new values
    SampleAlts = c(SampleAlts, alt)
    Types = c(Types, type)
    Places = c(Places, place)

  }

  #put all these into Master
  Master = cbind(CaseIDs, Types, Places, SampleAlts)
  Master = as.data.frame(Master)
  colnames(Master) = c('CaseIDs', 'Class', 'Location', 'Alteration')
  return(Master)
}

filterMaster <- function(Master, rna) {

  #get the case IDs from your RNA list
  rnaCaseIDs = rna[, 3]

  #check to see that all entries in Master are in the list
  out = c()
  for (x in 1:nrow(Master)) {
    if (Master$CaseIDs[x] %in% rnaCaseIDs) {

      out = c(out, T)

    } else {

      out = c(out, F)

    }
  }

  #Take out the entries that do not have accompanying RNA
  Master = Master[out, ]
  return(Master)
}

addPrimarySite <- function(Master) {

  #record the caseIDs in Master
  caseIDs = unique(Master$CaseIDs)

  #find the clinical data associated with those IDs
  clin = readRDS('<case IDS and project list>')
  clin = clin[clin$caseIDs %in% caseIDs,]

  #Make a new column in Master with the case IDs
  Master$Cancer = unlist(lapply(Master$CaseIDs, function(x) {

    return(clin$project[which(clin$caseIDs == x)])

  }))

  #remove heme cancers
  out = c('TCGA-THYM', 'TCGA-DLBC', 'TCGA-LAML', 'TCGA-LCML')
  Master = Master[!(Master$Cancer %in% out),]

  return(Master)
}

addSex <- function(Master) {

  #record the caseIDs in Master
  caseIDs = unique(Master$CaseIDs)

  #find the clinical data associated with those IDs
  clin = readRDS('<TCGA case IDs and biological sex>')
  clin = clin[clin$V1 %in% caseIDs,]

  #Make a new column in Master with the case IDs
  Master$Sex = unlist(lapply(Master$CaseIDs, function(x) {

    return(clin$V2[which(clin$V1 == x)])
  }))

  Master$Sex[is.na(Master$Sex)] = 'Unknown'
  return(Master)
}

getMaster <- function(input_dir, gene, rna) {
  #create first Master dataframe
  Master <- makeMaster(input_dir, gene)

  #filter Master to contain only samples with RNA available
  Master <- filterMaster(Master, rna)

  #Add primary site information to master
  Master <- addPrimarySite(Master)
  Master <- addSex(Master)

  # for genes on the X and Y chromosome there are different rules for men v women
  sex_genesY = readRDS('<genes on Y chromosome>')
  sex_genesX = readRDS('<genes on X chromosome>')

  #annotate the copy number alterations
  Master$Alteration = unlist(lapply(1:nrow(Master), function(r) {

    # record the sex and number of copies
    sex = Master$Sex[r]
    alt = Master$Alteration[r]

    if (T %in% grepl('CNV', alt)) {

      # extract the copies
      alt = gsub('CNV: ', "", alt)
      alt = as.numeric(alt)

      if (alt >= 6) {

        # if there are more than 6 copies, its a gain
        return('Gain')

      } else if (alt < 2) {

        # if there are less than 2 copies, we have to check the chromosome status
        if (sex == 'Unknown') {
          if (gene %in% c(sex_genesY, sex_genesX)) {

            # if we don't know the sex, and the gene on on the x or y chromosome,
            # we cannot determine anything
            return(NA)

          } else {

            # if the gene is not on X or Y it doesn't matter if we know the sex
            # of the patient
            return('Loss')

          }

        } else {

          if (sex == 'female') {
            if (gene %in% sex_genesY) {

              # if a woman has no copies of a Y chromsome gene, that's not a GA
              return(NA)

            } else {

              return('Loss')

            }

          } else {

            if (gene %in% sex_genesX) {

              # however if the patient is male, they need less than 1 copy of an
              # X chromosome gene
              return(ifelse(alt < 1, 'Loss', NA))

            } else {

              return('Loss')

            }
          }
        }

      } else {

        return(NA)

      }
    } else {

      return(alt)

    }
  }))

  #put together the data and return Master
  Master = Master[!(is.na(Master$Alteration)),]
  Master = Master[, colnames(Master) != 'Sex']
  Master = Master[Master$Alteration != "",]
  return(Master)
}
```

```
# In getControlRNA, we determine a list of samples NOT present in Master and
# then extract the batch corrected RNA seq files into a list of dataframes
# divided by cancer type
# each row of each dataframe is a gene
# each column of each dataframe is a sample

getControlRNA <- function(Master, rna, proj) {

  # extract WT sample IDs
  fileIDs = rna[!(rna[, 3] %in% Master$CaseIDs), ]

  # filter the IDs for the given cancer type
  fileIDs = rna[rna$Cancer == proj,]

  # remove any duplicates
  fileIDs = fileIDs[!(duplicated(fileIDs$caseIDs)),]

  #get the file names
  names = fileIDs$caseIDs
  fileIDs = paste0(fileIDs$Names, '_batch_corrected.csv')

  #create dataframe with those RNA files and add row/column names
  ControlRNA = lapply(fileIDs, function(x)
    fread(
      paste0('/fs/ess/PAS0854/Active_projects/TCGA_BatchResult/', x),
      select = 'x'
    )
  )
  ControlRNA = do.call(cbind, ControlRNA)
  colnames(ControlRNA) = names
  genes = readLines('/fs/ess/PAS0854/Active_projects/Gene_Names_TCGABatch.txt')
  ControlRNA = cbind(genes, ControlRNA)
  colnames(ControlRNA)[1] = 'gene_symbol'
  ControlRNA = as.data.frame(ControlRNA)
  rownames(ControlRNA) = ControlRNA[, 1]
  ControlRNA = ControlRNA[, -1]

  #return the data
  return(ControlRNA)
}

# In getMasterRNA, we determine a list of samples ARE present in Master and then
# extract the batch corrected RNA seq files into a dataframe
# each row of the dataframe is a gene
# each column of the dataframe is a sample

getMasterRNA <- function(Master, rna) {

  #get rna case IDs that are available
  fileIDs = rna[rna[, 3] %in% Master$CaseIDs, ]

  #get the file names
  names = fileIDs$caseIDs
  fileIDs = paste0(fileIDs$Names, '_batch_corrected.csv')

  #read in all the rna files
  MasterRNA = lapply(fileIDs, function(x)
    fread(
      paste0('/fs/ess/PAS0854/Active_projects/TCGA_BatchResult/', x),
      select = 'x'
    ))
  #put them together in a data frame
  MasterRNA = do.call(cbind, MasterRNA)
  colnames(MasterRNA) = names
  genes = readLines('/fs/ess/PAS0854/Active_projects/Gene_Names_TCGABatch.txt')
  MasterRNA = cbind(genes, MasterRNA)
  colnames(MasterRNA)[1] = 'gene_symbol'
  MasterRNA = as.data.frame(MasterRNA)
  rownames(MasterRNA) = MasterRNA[, 1]
  MasterRNA = MasterRNA[, -1]

  #return data
  return(MasterRNA)
}

# In makeRNAright we normalize the gene counts and convert the count dataframe
# to TPM to keep consistent with the data format from our model building

makeRNAright <- function(rna) {

  # create a pseudo annotation dataframe to normalize data
  coldata = rep('mutant', ncol(rna))
  coldata = as.data.frame(coldata)
  colnames(coldata) = c('condition')
  coldata$condition = as.factor(coldata$condition)

  # normalize counts
  t_rna = DESeqDataSetFromMatrix(countData = rna,
                                 colData = coldata,
                                 design = ~ 1)
  t_rna <- estimateSizeFactors(t_rna)
  t_rna = counts(t_rna, normalized = T)

  # read in gene lengths
  geneLen = readRDS('<gene lengths>')

  # match the order of gene lengths to the order of the RNA
  geneLen = geneLen[geneLen$Gene_Symbol %in% rownames(t_rna),]
  t_rna = t_rna[rownames(t_rna) %in% geneLen$Gene_Symbol,]
  geneLen = geneLen[match(rownames(t_rna), geneLen$Gene_Symbol),]

  # convert rna to TPM
  t_rna = convertCounts(as.matrix(t_rna), 'TPM', geneLen$Length)
  t_rna = as.data.frame(t_rna)

  #return the RNA
  return(t_rna)
}

# In addFunction and addImmuneProb, we add a column to Master which has the
# function or immune phenotype respectively of the sample in which the
# alteration is present.

addFunction <- function(Master, Mutant_Function) {

  #Make sure cases match
  Mutant_Function = Mutant_Function[Mutant_Function[, 1] %in% Master$CaseIDs,]

  #create new column function made of case IDs
  Master$Function = Master$CaseIDs

  #use Mutant_Function as a key to replace case IDs with functions
  for (i in 1:nrow(Mutant_Function)) {
    Master$Function = replace(Master$Function,
                              which(Master$Function == Mutant_Function[i, 1]),
                              Mutant_Function[i, 2])
  }

  return(Master)
}

addImmuneProb <- function(Master, ImmuneProb) {

  # check the immuneProb you made matches
  ImmuneProb = ImmuneProb[ImmuneProb$Names %in% Master$CaseIDs,]

  #create new column function made of case IDs
  Master$Immune = Master$CaseIDs

  #use ImmuneProb as a key to replace case IDs with functions
  for (i in 1:nrow(ImmuneProb)) {

    #replace each caseID with the immune phenotype
    id = which(Master$Immune == ImmuneProb[i, 1])
    Master$Immune[id] = ImmuneProb[i, 2]

  }

  return(Master)
}

# In check conditions we ensure the mutation is evaluable
# In getAltImmuneTesting we test each mutant group's immune
# phenotypes within the same cancer type
# In getAltFunctionTesting we test each mutants group's functional
# phenotypes overall
# In getAltFunctionImmuneTesting we run the previous functions an tally the
# results together in a reasonable table
```

```r
checkConditions <- function(splitedMaster) {

  # check that there are at least 5 instances of the alteration in the cancer type
  # check that all the function annoatations are not "unknown"
  out = c(nrow(splitedMaster) >= 5,
          T %in% (splitedMaster$Function != 'Unknown'))

  if (F %in% out) {

    return(NA)

  } else {

    return(splitedMaster)

  }
}

getAltImmuneTesting <- function(Master, props) {

  # split master based on alteration and cancer type
  master.split <-
    split(Master, paste0(Master$Alteration, "_", Master$Cancer))

  # check that each combination is evaluable
  master.split <- lapply(master.split, checkConditions)

  # remove unevaluable alterations
  master.split = master.split[!(is.na(master.split))]

  if (length(master.split) == 0) {

    # if nothing is evaluavkle
    # resplit
    master.split <-
      split(Master, paste0(Master$Alteration, "_", Master$Cancer))

    alts = names(master.split)
    nHots = c()
    nColds = c()
    totals = c()
    sigsI = c()
    im = c()
    # record number of each group (GOF, LOF, High, Low)

    for (x in alts) {

      tmp = master.split[[x]]
      total = nrow(tmp)
      nHot = nrow(tmp[tmp$Immune == 'High',])
      nCold = nrow(tmp[tmp$Immune == 'Low',])
      nHots = c(nHots, nHot)
      nColds = c(nColds, nCold)
      sigsI = c(sigsI, 1)
      im = c(im, 'Unknown')
      totals = c(totals, total)

    }

  } else {

    # if there is at least 1 evaluable alteration
    alts = names(master.split)
    nHots = c()
    nColds = c()
    totals = c()
    sigsI = c()
    im = c()

    # go through each alteration that you can evaluate
    for (x in alts) {

      # store data for manipulation
      tmp = master.split[[x]]

      # record the total number of samples
      total = nrow(tmp)

      # record the porportions of high and low sampes in the parent cancer type
      props_can = props[props$Cancer == tmp$Cancer[1],]
      prop_high = props_can[1, 3] / props_can[1, 2]
      prop_low = props_can[1, 4] / props_can[1, 2]

      #if any are 0 or 1, offset by .01 to allow for evaluation
      prop_high = ifelse(prop_high == 0, .01, prop_high)
      prop_low = ifelse(prop_low == 1, .99, prop_low)

      # record number of each group (High, Low)
      nHot = nrow(tmp[tmp$Immune == 'High',])
      nCold = nrow(tmp[tmp$Immune == 'Low',])

      # determine if the proportion of hot samples is greater than expected
      z = prop.test(
        nHot,
        total,
        p = prop_high,
        alternative = "greater",
        correct = TRUE
      )

      if (z$p.value < .05) {

        # if it is greater than expected record the number of highs and lows
        totals = c(totals, total)
        nHots = c(nHots, nHot)
        nColds = c(nColds, nCold)
        # record the pvalue
        sigsI = c(sigsI, z$p.value)

        # record "high" annoation
        im = c(im, 'High')

      } else {

        # if there is not more highs than expected, test lows
        z = prop.test(
          nCold,
          total,
          p = prop_low,
          alternative = "greater",
          correct = TRUE
        )

        if (z$p.value < .05) {

          # if there are more lows than expected record the number of highs and lows
          totals = c(totals, total)
          nHots = c(nHots, nHot)
          nColds = c(nColds, nCold)

          # record the pvalue
          sigsI = c(sigsI, z$p.value)

          # record "high" annoation
          im = c(im, 'Low')

        } else {

          # if there is no difference in lows either record everything
          totals = c(totals, total)
          nHots = c(nHots, nHot)
          nColds = c(nColds, nCold)
          sigsI = c(sigsI, z$p.value)

          # record the annotation unknown
          im = c(im, 'Unknown')

        }
      }
    }
  }
  # collect all the annotations
  alt <- cbind(alts, totals, nHots, nColds, sigsI, im)
  colnames(alt) = c('Alt.ID',
                    'Total.Samples',
                    'nHighs',
                    'nLows',
                    'Immu.Sig',
                    'Immune')
  # return them all in a dataframe
  alt = as.data.frame(alt)

  return(alt)
}

getAltFunctionTesting <- function(Master) {

  # split master based on alteration
  master.split <- split(Master, Master$Alteration)

  # check that each combination is evaluable
  master.split <- lapply(master.split, checkConditions)
  master.split = master.split[!(is.na(master.split))]

  if (length(master.split) == 0) {

    # if nothing is evaluavkle
    # resplit
    master.split <- split(Master, Master$Alteration)
    alts = names(master.split)
    nGOFs = c()
    nLOFs = c()
    sigsF = c()
    fun = c()

    for (x in alts) {

      # record number of each group (GOF, LOF, High, Low)
      tmp = master.split[[x]]
      nGOF = nrow(tmp[tmp$Function == 'GOF', ])
      nLOF = nrow(tmp[tmp$Function == 'LOF', ])
      nGOFs = c(nGOFs, nGOF)
      nLOFs = c(nLOFs, nLOF)
      sigsF = c(sigsF, 1)
      fun = c(fun, 'Unknown')

    }
  } else {

    # if there is at least 1 evaluable alteration
    alts = names(master.split)
    nGOFs = c()
    nLOFs = c()
    sigsF = c()
    fun = c()

    # go through each alteration that you can evaluate
    for (x in alts) {

      # record the number of GOFs and LOFs
      tmp = master.split[[x]]
      nGOF = nrow(tmp[tmp$Function == 'GOF', ])
      nLOF = nrow(tmp[tmp$Function == 'LOF', ])

      # determine if there are more or less GOFs than expected by random chance
      z = prop.test(
        nGOF,
        sum(nLOF, nGOF),
        p = NULL,
        alternative = "two.sided",
        correct = TRUE
      )

      if (z$p.value < .05) {

        # if there are an unexpected number of GOFs
        if (nGOF > nLOF) {

          # if there are more GOFs than LOFs
          # record all the numbers
          nGOFs = c(nGOFs, nGOF)
          nLOFs = c(nLOFs, nLOF)
          sigsF = c(sigsF, z$p.value)

          # record the annotation GOF
          fun = c(fun, 'GOF')

        } else {

          # if there are more LOFs than GOFs
          # record all the numbers
          nGOFs = c(nGOFs, nGOF)
          nLOFs = c(nLOFs, nLOF)
          sigsF = c(sigsF, z$p.value)

          # record the annotation LOF
          fun = c(fun, 'LOF')

        }
      } else {

        # there are about as many GOFs we would expect
        # record all the numbers
        nGOFs = c(nGOFs, nGOF)
        nLOFs = c(nLOFs, nLOF)
        sigsF = c(sigsF, z$p.value)

        # record the annotation "unknown"
        fun = c(fun, 'Unknown')

      }
    }
  }

  # record all the values together
  alt <- cbind(alts, nGOFs, nLOFs, sigsF, fun)
  colnames(alt) = c('Alt.ID', 'nGOFs', 'nLOFs', 'Func.Sig', 'Function')

  # return the data together
  alt = as.data.frame(alt)
  return(alt)
}

getAltFunctionImmuneTesting <- function(Master, props) {

  # get the function results
  alt.data.f = getAltFunctionTesting(Master)

  # get the immune results
  alt.data.i = getAltImmuneTesting(Master, props)

  # colect all the function results for each alteartion in the immune results
  out = lapply(gsub("\\_TCGA.*", "", alt.data.i$Alt.ID), function(x) {

    id = which(alt.data.f$Alt.ID == x)
    return(alt.data.f[id,])

  })

  # combine all the results together
  out = do.call(rbind, out)
  alt.data = cbind(alt.data.i, out[,-1])

  # make the column names be a particular order
  alt.data = alt.data[, match(
    c(
      'Alt.ID',
      'Total.Samples',
      'nGOFs',
      'nLOFs',
      'Func.Sig',
      'Function',
      'nHighs',
      'nLows',
      'Immu.Sig',
      'Immune'
    ),
    colnames(alt.data)
  )]

  # return the resulting data frame
  return(alt.data)
}

getAltFunctionImmuneFinal <- function(alt) {

  # some of the rows from the alt produced by getAltFunctionImmuneTesting
  # have unnecessary rows
  # filter to only include the needed rows
  alt = data.frame(
    Alt.ID = alt[1, 1],
    Total.Samples = sum(as.numeric(alt[, 2])),
    nGOFs = sum(as.numeric(alt[, 3])),
    nLOFs = sum(as.numeric(alt[, 4])),
    Func.Sig = mean(as.numeric(alt[, 5])),
    Function = alt[1, 6],
    nHighs = sum(as.numeric(alt[, 7])),
    nLows = sum(as.numeric(alt[, 8])),
    Immu.Sig = mean(as.numeric(alt[, 9])),
    Immune = alt[1, 10],
    Cancer = paste0(alt[, 11], collapse = ", ")
  )

  return(alt)
}

#In filterResults we create the necesary dataframes for the graphs created in R shiny.

filterResults <-
  function(Master,
           MasterRNA,
           ControlRNA,
           Cancers = 'all',
           Mutations = 'all') {

    #record the caseIDs in ControlRNA
    caseIDs = colnames(ControlRNA)

    #find the clinical data associated with those IDs
    clin = readRDS('<clinical data>')

    # filter clin to only cinlude the data in ControlRNA
    clin = clin[clin$caseIDs %in% caseIDs,]
    clin = clin[match(colnames(ControlRNA), clin$caseIDs),]

    # store the cancer types in ControlCan
    ControlCan = as.data.frame(cbind(colnames(ControlRNA), clin$project))
    colnames(ControlCan) = c('CaseIDs', 'Cancers')

    if (Cancers != 'all') {

      # if the cancer parameter is changed, filter accordingly
      Master = Master[Master$Cancer %in% Cancers,]
      MasterRNA = MasterRNA[, colnames(MasterRNA) %in% Master$CaseIDs]
      ControlCan = ControlCan[ControlCan[, 2] %in% Cancers,]
      ControlRNA = ControlRNA[, colnames(ControlRNA) %in% ControlCan[, 1]]

    }
    if (Mutations != 'all') {

      # if the Mutations parameter is changed, filter accordingly
      Master = Master[Master$Alteration %in% Mutations,]
      MasterRNA = MasterRNA[, colnames(MasterRNA) %in% Master$CaseIDs]

    }
    if (ncol(ControlRNA) < 500) {

      # combine the RNA for graphing
      rna = cbind(ControlRNA, MasterRNA)

      # ensure there are no inadvertant duplicated rows
      funs = Master[!(duplicated(Master$CaseIDs)), 1]
      funs = funs[match(colnames(MasterRNA), funs)]

      # make mutant data metadata for function
      for (i in 1:nrow(Master)) {

        funs = replace(funs, which(funs == Master[i, 1]), Master[i, 6])

      }

      imm = Master[!(duplicated(Master$CaseIDs)), 1]
      imm = imm[match(colnames(MasterRNA), imm)]

      # make mutant data metadata for immune
      for (i in 1:nrow(Master)) {

        imm = replace(imm, which(imm == Master[i, 1]), Master[i, 7])

      }

      # make immune and function control RNA metadata for graphing
      metadata1 = c(rep('Control', ncol(ControlRNA)), funs)
      metadata2 = c(rep('Control', ncol(ControlRNA)), imm)

    } else {

      # if there are more than 500 samples in the control data, randomly
      # select 500 to preserve RAM on the web portal
      rna = cbind(ControlRNA[, sample(1:ncol(ControlRNA), 500)], MasterRNA)

      # ensure there are no inadvertant duplicated rows
      funs = Master[!(duplicated(Master$CaseIDs)), 1]
      funs = funs[match(colnames(MasterRNA), funs)]

      # make mutant data metadata for function
      for (i in 1:nrow(Master)) {

        funs = replace(funs, which(funs == Master[i, 1]), Master[i, 6])

      }
      imm = Master[!(duplicated(Master$CaseIDs)), 1]
      imm = imm[match(colnames(MasterRNA), imm)]
      # make mutant data metadata for immune
      for (i in 1:nrow(Master)) {

        imm = replace(imm, which(imm == Master[i, 1]), Master[i, 7])

      }

      # make immune and function control RNA metadata for graphing
      metadata1 = c(rep('Control', ncol(ControlRNA)), funs)
      metadata2 = c(rep('Control', ncol(ControlRNA)), imm)

    }

    #put together the meta data
    metadata = cbind(metadata1, metadata2, colnames(rna))
    metadata = as.data.frame(metadata)
    colnames(metadata) = c('Function', 'Immune', 'IDs')

    # put together all the data for the final product
    l = list(rna, metadata, Master, ControlRNA, MasterRNA, ControlCan)
    names(l) = c('rna',
                 'metadata',
                 'Master',
                 'ControlRNA',
                 'MasterRNA',
                 'ControlCan')

    return(l)
  }

# in runRIGATONI we perform all the functions of RIGATONI and all
# the parts of the TCGA analysis.

runRIGATONI <- function(gene, input_dir) {

  # read in all genes in the TCGA RNA
  all_genes = readLines('<TCGA genes>')

  if (gene %in% all_genes) {

    # if the gene you are trying to evaluate is in the list
    # remove the list of genes
    rm(all_genes)
    message(paste0('Starting gene: ', gene))
    message('Setting up')

    # load the RNA master list and the string database data
    sdb <- readRDS('<string data>')
    rna <- readRDS('<rna available>')

    #make the master dataframe
    message('Making master dataframe for mutant samples')
    Master <- suppressWarnings(getMaster(input_dir, gene, rna))
    Master = Master[!(duplicated(Master$CaseIDs)),]

    # get the control RNA
    message('Gathering wild-type control RNA')
    ControlRNA <-
      suppressMessages(lapply(unique(Master$Cancer), function(x) {
        c = getControlRNA(Master, rna, x)
        return(makeRNAright(c))
      }))
    names(ControlRNA) = unique(Master$Cancer)

    #get the mutant RNA
    message('Gathering mutant RNA')
    MasterRNA <- suppressMessages(getMasterRNA(Master, rna))
    MasterRNA = makeRNAright(MasterRNA)
    rm(rna)

    # make gene list
    message('Making Initial Gene List')
    gene_list_ppi = makeGeneList(gene, sdb)

    if (length(unlist(gene_list_ppi)) > 0) {

      rm(sdb)

      # build the regression model
      message('Building regression model')

      Regression <- lapply(ControlRNA, function(x) {

        # build each regression within each cancer type
        suppressWarnings(getRegression(gene_list_ppi, gene, x))

      })
      names(Regression) <- names(ControlRNA)

      # predict mutant function
      message('Predicting mutant function')
      Mutant_Function <- lapply(unique(Master$Cancer), function(x) {

        # predict mutant function within each cancer type individually
        # create psuedo Master_1 for evaluation
        Master_1 = Master[Master$Cancer == x,]

        # create psuedo MasterRNA_1 for evaluation
        MasterRNA_1 = MasterRNA[, colnames(MasterRNA) %in% Master_1$CaseIDs]

        # extract the control regression matching the cancer type
        Regression_1 = Regression[[x]]

        # determine mutant function
        out = tryCatch({
```

```
      return(suppressWarnings(getMutantFunction(
        Regression_1, MasterRNA_1, gene
      )))

    }, error = function(e) {

      return(cbind(Master_1$CaseIDs, 'Unknown'))

    })

    return(out)
  })

  # put together all the mutant functions
  Mutant_Function = do.call(rbind, Mutant_Function)

  # add the mutant functions to master
  Master <- addFunction(Master, Mutant_Function)

  # predict immune phenotype
  message('Predicting immune phenotype')
  ImmuneProb <- suppressMessages(getImmuneProb(MasterRNA, m))
  Master <- suppressMessages(addImmuneProb(Master, ImmuneProb))

  # predict alteration level data
  message('Predicting alteration function and immune phenotype')

  # get pre-recorded cancer type immune phenotype proportions
  props = readRDS('/fs/ess/PAS0854/Raven/immune-genomics/TCGA_Proportions.RDS')

  # get the initial alt data frame
  alt.data <-
    suppressWarnings(getAltFunctionImmuneTesting(Master, props))
  alt.data = as.data.frame(alt.data)

  # keep raw results in alt.data.keep for record keeping
  alt.data.keep = alt.data
  alt.data.keep$gene = gene

  # filter alt.data for interesting results
  alt.data = alt.data[alt.data$Function != 'Unknown',]
  alt.data = alt.data[alt.data$Immune != 'Unknown',]

  if (nrow(alt.data) == 0) {

    # if alt.data is empty
    # just keep alt.data.keep
    message(
      paste0(
        gene,
        ' has no functional, immune-affecting alterations after filtering'
      )
    )
```

```
        l = list(alt.data.keep)

        names(l) = c('alt.data.keep')

        return(l)

      } else {

        # if alt.data is not empty
        message('Getting final output')

        # filter Master so it just has the positive results
        Master = Master[paste0(Master$Alteration, "_", Master$Cancer) %in%
                          alt.data$Alt.ID,]

        # add a cancer column
        alt.data$Cancer = unlist(lapply(alt.data$Alt.ID, function(x) {
          out = strsplit(x, "_")[[1]][2]
        }))

        # fix the alt.ID column to not contain cancer information now in cancer column
        alt.data$Alt.ID = unlist(lapply(alt.data$Alt.ID, function(x) {
          out = strsplit(x, "_")[[1]][1]
        }))

        # for each group of cancers which behaves the same, separate them into
        # sub-data frames
        alt.data = split(
          alt.data,
          paste0(
            alt.data$Alt.ID,
            "_",
            alt.data$Function,
            "_",
            alt.data$Immune
          )
        )

        # summarize each sub-dataframe
        alt.data = lapply(alt.data, function(x)
          return(getAltFunctionImmuneFinal(x)))

        # combine all summarized data
        alt.data = do.call(rbind, alt.data)
        rownames(alt.data) = NULL

        # filter MasterRNA to include only data from interesting results
        MasterRNA = MasterRNA[, colnames(MasterRNA) %in% Master$CaseIDs]

        # remove cancer types with no interesting results from ControlRNA
        ControlRNA = ControlRNA[names(ControlRNA) %in% Master$Cancer]

        # combine ControlRNA into one dataframe
        ControlRNA = do.call(cbind, ControlRNA)

        colnames(ControlRNA) = unlist(lapply(colnames(ControlRNA), function(x) {

          # remove cancer types from column names
          x = strsplit(x, "\\.")[[1]][2]

          return(x)

        }))

        # create the final output
        forGraphs <-
          suppressMessages(filterResults(Master, MasterRNA, ControlRNA))

        l = list(
          Master,
          MasterRNA,
          ControlRNA,
          alt.data,
          forGraphs[['rna']],
          forGraphs[['metadata']],
          forGraphs[['ControlCan']],
          alt.data.keep
        )

        message('Done!')

        names(l) = c(
          'Master',
          'MasterRNA',
          'ControlRNA',
          'alt.data',
          'rna',
          'metadata',
          'ControlCan',
          'alt.data.keep'
        )

        # return
        return(l)
      }
    } else {
      message(paste0(gene, ': no mapped genes :('))
      return(NA)
    }
  } else {
    message(paste0(gene, " Not in gene list"))
    return(NA)
  }
}
```

6.2: Run RIGATONI for all genes

In order to run RIGATONI for all genes, I created a plain text file with a new gene to run on each line of the file and two batch scripts to run each gene through my pipeline in parallel.

In the first batch script, I run each batch job individually and ensure I do not excede batch limits.

```
#!/bin/sh
#SBATCH --account=PAS0854
#SBATCH --time=30:00:00
#SBATCH --ntasks=5

input_dir='/fs/ess/PAS0854/Active_projects/TCGA_novel_icktps'
output_dir='/fs/scratch/PAS0854/vella12/RIGATONI_Shiny'

while IFS= read line;
do
  sbatch Evaluation.sh $line `echo $input_dir` `echo $output_dir`
  cond=`squeue -u vella12 | grep 'Val' | wc -l`
  while [ $cond -gt 998 ];
  do
    echo "Waiting";
    sleep 10m;
    cond=`squeue -u vella12 | grep 'Val' | wc -l`;
  done
  done < <TCGA gene names>
```

In the next batch script, I simply run the evaluation script in R described in detail in 5.1

```
#!/bin/sh
#SBATCH --account=PAS0854
#SBATCH --time=30:00:00
#SBATCH --ntasks=10

module load R/4.1.0-gnu9.1

Rscript evaluationUsingGenes.R $1 $2 $3
```

The evaluationUsingGenes.R script simply loads each function described in 5.1, and the R package described in 4 and then executes the following code:

```
# read in the command line arguments
args = commandArgs(trailingOnly = T)
gene = as.character(args[1])
gene = gsub('[\r\n]', '', gene)
input_dir = as.character(args[2])
output_dir = as.character(args[3])

# run RIGATONI
RIGA <- runRIGATONI(gene, input_dir)

# check the RIGATONI ran all the way through
if (class(RIGA[[1]]) == 'data.frame') {
  if (length(RIGA) == 1) {

    # if there is only 1 element, that means there were no important alterations
    # keep the record

    write.csv(
      RIGA$alt.data.keep,
      paste0(
        '/fs/ess/PAS0854/Raven/all.alt.data/',
        gene,
        '.alt.data.csv'
      )
    )

  } else if (length(RIGA) > 1) {
    # if there are more than 1 elements, that means it ran to completion
    # keep the record

    write.csv(
      RIGA$alt.data.keep,
      paste0(
        '/fs/ess/PAS0854/Raven/all.alt.data/',
        gene,
        '.alt.data.csv'
      )
    )

    # remove the record
    RIGA = RIGA[1:7]

    #create directory for the results
    dir.create(paste0(output_dir, "/", gene))

    # go to the directory and save the object
    setwd(paste0(output_dir, "/", gene))
    saveRDS(object = RIGA,
            file = 'RIGA.rdata',
            compress = T)
  }
}
```

Part 7: Text mining (Fig 4)

We extracted all the results from the Pan TCGA analysis and then performed text mining to understand how our results compare to existing literature. First we ran the following in linux.

```
esearch -db pubmed -query "(((immunity) OR (immunology) OR (immune)) \
AND (cancer) NOT (Review[Publication Type]) NOT (Preprint[Publication Type])) \
AND (("2010/12/6"[Date - Publication] : "3000"[Date - Publication]))" |
efetch -format abstract >> abstracts.test.txt
```

Next we did the following in R

```r
# read in the abstracts file created before
abstracts <- readabs('abstracts.test.txt')

# 226093 total papers analyzed
# annotate how many times each gene appears in the abstract object
genes = gene_atomization(abstracts)

# convert to a data frame
genes = data.frame(genes)

# save for future use
saveRDS(genes, 'PubMedGenes.RDS')

# go to the location of all the recording keeping files stored in section 6
riga_visualize = list.files('/fs/ess/PAS0854/Raven/all.alt.data/')

# read each file
riga_visualize = lapply(riga_visualize, read.csv)

# combine all the files together
riga_visualize = do.call(rbind, riga_visualize)

#filter for only interesting results
riga_visualize = riga_visualize[riga_visualize$Function != 'Unknown', ]
riga_visualize = riga_visualize[riga_visualize$Immune != 'Unknown', ]

# filter the pubmed genes to include only genes in our output
genes = genes[genes$Gene_symbol %in% riga_visualize$gene, ]
riga_visualize$PubMedref = unlist(lapply(riga_visualize$gene, function(gene) {

  # for each gene
  if (gene %in% genes$Gene_symbol) {

    id = which(genes$Gene_symbol == gene)

    #record the number of times the gene appeared
    return(genes$Freq[id])

  } else {

    # if it did not appear, record 0
    return(0)

  }
}))

# remove any duplicated values
riga_visualize = riga_visualize[!(duplicated(riga_visualize$gene)), ]

# make sure the number of references is numeric
riga_visualize$PubMedref = as.numeric(as.character(riga_visualize$PubMedref))

# convert to log10
```

```
riga_visualize$PubMedref = log10(riga_visualize$PubMedref)

# if the value is non-finite after converting with log10, convert to 0
riga_visualize$PubMedref[!(is.finite(riga_visualize$PubMedref))] = 0

# Figure 4A
p1 = ggplot(riga_visualize, aes(x = PubMedref)) +
  geom_histogram(
    data = riga_visualize,
    color = 'black',
    fill = 'gray',
    binwidth = 1,
    alpha = .5
  ) +
  geom_text(
    aes(label = ifelse(after_stat(count) > 0, after_stat(count))),
    y = c(2880, 2200, 900, 150, 80),
    stat = "bin",
    binwidth = 1,
    size = 5
  ) +
  theme_classic() +
  scale_x_continuous(
    breaks = c(0, 1, 2, 3, 4),
    labels = c("0", '1-10', '11-100', '101-1,000', '1,000-10,000'),
    expand = c(.04, .04)
  ) +
  scale_y_continuous(expand = expansion(mult = c(.01, .08))) +
  xlab('PubMed References Connecting \nGOI to Cancer Immunity') +
  ylab('Number of Genes (n=5746)') +
  theme(text = element_text(size = 15),
        axis.text.x = element_text(
          angle = 45,
          vjust = 1,
          hjust = 1
        ))
saveRDS(p1, "<Figure 4A>")
```

Part 8: 14q deletion in renal cell carcinoma (Fig 4)

All data shown here is from the paper found at "https://aacrjournals.org/cancerres/article/83/5/700/716683/Integrative-Single-Cell-Analysis-Reveals" The bulk RNAseq results for 14q deletions shown in figures 4B and 4C are extracted from the pan-TCGA analysis described in section 6. The single cell analysis is shown here. All data was downloaded from "https://www.ncbi.nlm.nih.gov/geo/query/acc.cgi?acc=GSE207493"

8.1 Preprocessing

```
# this function reads in all the data and performs quality control filtering
read_data <- function(RCC) {
  #read the data in
  out.data <- Read10X(data.dir = paste0("./", RCC))

  #turn the file into a Seurat object
  out <-
    CreateSeuratObject(
      counts = out.data,
      project = "mRCC81",
      min.cells = 8,
      min.features = 200
    )

  # record the percent of mitochondrial genes
  out[["percent.mt"]] <- PercentageFeatureSet(out, pattern = "^MT-")

  #record the lower bound of the feature count
  low = median(out$nFeature_RNA) - 5 * (sd(out$nFeature_RNA))

  if (low < 0) {
    # if the calculated lower bound is below 0, set to 200
    low = 200
  }

  # calculate the upper bound of feature count
  high = median(out$nFeature_RNA) + 5 * (sd(out$nFeature_RNA))


  if (high > max(out$nFeature_RNA)) {
    # if the upper bound is higher than the max, set the max to the max + 1
    high = max(out$nFeature_RNA) + 1
  }

  # calculate the upper bound of counts
  high_c = median(out$nCount_RNA) + 5 * (sd(out$nCount_RNA))

  if (high_c > max(out$nCount_RNA)) {
    # if the upper bound is higher than the max, set the max to the max + 1
    high_c = max(out$nCount_RNA) + 1
  }

  # remove cells outside the calculated bounds and
  # with percent.mt greater than or equal to 10
  out <-
    subset(out,
           subset = nFeature_RNA > low &
             nFeature_RNA < high & percent.mt < 10 & nCount_RNA < high_c)

  #set the orig.ident to the sample name
  out$orig.ident = factor(rep(RCC, length(out$orig.ident)), levels = c(RCC))

  #return the object
  return(out)
}
```

```
# list all the samples to study
ls = list.dirs()

# put them into the correct format
ls = gsub('\\./', "", ls)

# get just the RCC samples
ls = ls[grepl('RCC', ls)]

# read in all the data
mRcc = lapply(ls, read_data)

# merge together all the experiments
mRcc = merge(x = mRcc[[1]], y = mRcc[2:length(mRcc)])
mRcc[["joined"]] <- JoinLayers(mRcc[["RNA"]])

# check the quality control filtering
mRcc[["percent.mt"]] <- PercentageFeatureSet(mRcc, pattern = "^MT-")
VlnPlot(
  mRcc,
  features = c("nFeature_RNA", "nCount_RNA", "percent.mt"),
  ncol = 3,
  pt.size = 0
)

# record the genes for the S and G2M phases of the cell cycle
s.genes <- cc.genes$s.genes
g2m.genes <- cc.genes$g2m.genes

# this is based on a tutorial from https://satijalab.org/seurat/articles/
# cell_cycle_vignette
# normalize and predict malignant cells
mRcc <- NormalizeData(mRcc, assay = 'joined')
mRcc <- FindVariableFeatures(mRcc, selection.method = "vst", nfeatures = 2000,
                             assay = 'joined')
mRcc <- CellCycleScoring(mRcc, s.features = s.genes,
                   g2m.features = g2m.genes,
                   set.ident = TRUE,
                   assay = 'joined')
mRcc <- ScaleData(mRcc, vars.to.regress = c("S.Score", "G2M.Score"),
                  assay = 'joined')

# set the default assay to the joined data
DefaultAssay(mRcc) = 'joined'

# run PCA
mRcc <-
  RunPCA(
    mRcc,
    pc.genes = VaribaleFeatures(mRcc, assay = 'joined'),
    npcs = 30,
    verbose = FALSE
  )
```

```
# explore best dimensions for clustering with a heatmap
DimHeatmap(mRcc,
           dims = 1:15,
           cells = 500,
           balanced = TRUE)

# explore best dimensions for clustering with a elbow plot
ElbowPlot(mRcc)
```

8.2 Cell typing for normal and tumor groups

The following is based on tutorials from "https://hbctraining.github.io/scRNA-seq_online/lessons/06a_integration_harmony.html" and "https://github.com/IanevskiAleksandr/sc-type"

```
mRcc <- mRcc %>%
  # harmonize to remove batch effect
  RunHarmony("orig.ident", plot_convergence = TRUE)

# find optimal clustering
resolution.range <- seq(from = 0, to = 1, by = 0.2)

# use the dimensions selected from the elbow plot and heatmap in 8.1
mRcc <- mRcc %>%

  # run UMAP on the harmonized data
  RunUMAP(reduction = "harmony", dims = 1:7) %>%

  # find the nearest neighbors for the UMAP object
  FindNeighbors(reduction = "harmony", dims = 1:7) %>%

  # find the clusters of the UMAP object across the resolution range
  FindClusters(resolution = resolution.range) %>%
  identity()

# display the clustree
clustree(mRcc, prefix = "joined_snn_res.")
# choose an ideal clustering based on probabilities

Idents(mRcc) = mRcc$joined_snn_res.0.4

rm(g2m.genes)
rm(ls)
rm(resolution.range)
rm(s.genes)

smRcc.markers <- FindAllMarkers(mRcc, only.pos = TRUE)

# load gene set preparation function
source(
  "https://raw.githubusercontent.com/IanevskiAleksandr/sc-type/master/R
  /gene_sets_prepare.R"
)
```

```r
# load cell type annotation function
source(
  "https://raw.githubusercontent.com/IanevskiAleksandr/sc-type/master/R
  /sctype_score_.R"
)
# DB file
db_ <-
  "https://raw.githubusercontent.com/IanevskiAleksandr/sc-type/master
/ScTypeDB_full.xlsx"

tissue <- "Kidney"

# prepare gene sets
gs_list <- gene_sets_prepare(db_, tissue)

# check Seurat object version (scRNA-seq matrix extracted differently in Seurat v4/v5)
seurat_package_v5 <-
  isFALSE('counts' %in% names(attributes(mRcc[["joined"]])))

# extract scaled scRNA-seq matrix
scRNAseqData_scaled <-
  if (seurat_package_v5) {
    as.matrix(mRcc[["joined"]]$scale.data)
  } else {
    as.matrix(mRcc[["joined"]]@scale.data)
  }


# run ScType
es.max <-
  sctype_score(
    scRNAseqData = scRNAseqData_scaled,
    scaled = TRUE,
    gs = gs_list$gs_positive,
    gs2 = gs_list$gs_negative
  )

# merge by cluster
cL_resutls <-
  do.call("rbind", lapply(unique(mRcc@meta.data$joined_snn_res.0.4),
                          function(cl) {
    es.max.cl = sort(rowSums(es.max[, rownames(mRcc@meta.data[
      mRcc@meta.data$joined_snn_res.0.4 == cl,])
      ]),
      decreasing = !0)
    head(data.frame(
      cluster = cl,
      type = names(es.max.cl),
      scores = es.max.cl,
      ncells = sum(mRcc@meta.data$joined_snn_res.0.4 == cl)
    ),
    10)
  }))
```

```r
# collect the top cell type for each cluster
sctype_scores <-
  cL_resutls %>% group_by(cluster) %>% top_n(n = 1, wt = scores)
sctype_scores = as.data.frame(sctype_scores)

# there is one cluster that has a variety of scores close together,
# one of these scores being cancer stem cells
# based on the results, I believe these to be cancer stem cells and
# I'm manually changing that annotation
sctype_scores[8, ] = c(4, 'Cancer stem cells', 12590.768, 9015)

# set low-confident (low ScType score) clusters to "unknown"
sctype_scores$type[as.numeric(as.character(sctype_scores$scores)) <
                     as.numeric(as.character(sctype_scores$ncells)) /
                     4] <- "Unknown"

# add a new metadata slot which includes the sctype classification
mRcc@meta.data$sctype_classification = ""
for (j in unique(sctype_scores$cluster)) {
  cl_type = sctype_scores[sctype_scores$cluster == j, ]

  mRcc@meta.data$sctype_classification[mRcc@meta.data$joined_snn_res.0.4 == j]
  = as.character(cl_type$type[1])
}

# set the ident back to the sample level information
Idents(mRcc) = mRcc$orig.ident
```

8.3 Copy number analysis

```r
test = lapply(unique(mRcc$orig.ident), function(x) {

  print(x)

  # separate the sample you would like to work on
  out = subset(mRcc, idents  = x)

  # reset the Ident to the cell type
  Idents(out) = out$sctype_classification

  # collect the normal cells
  norm_names = names(out$sctype_classification[out$sctype_classification ==
                                                 'Hematopoietic cells'])

  # keep only the normal cells and tumor cells
  out = subset(out, idents  = c('Cancer stem cells', 'Hematopoietic cells'))

  # extract the RNA
  out = out@assays$joined$counts
  out = as.matrix(out)
  # keep the RNA and the normal names
  out = list(out, norm_names)
  names(out) = c('matrix', 'normal_names')
  return(out)

})

# save for future use
names(test) = unique(mRcc$orig.ident)
save(test, file = 'copyKatInput.rdata')

# run copykat
copykat.output = mclapply(1:length(test), function(x) {

  # record the sample name
  nam = names(test)[x]
  use = test[[x]]
  print(nam)

  # run copykat
  out = c(
    rawmat = use$matrix,
    id.type = "S",
    ngene.chr = 5,
    win.size = 25,
    KS.cut = 0.1,
    sam.name = nam,
    norm.cell.names = use$normal_names,
    distance = "euclidean",
    output.seg = "FLASE",
    plot.genes = "FALSE",
    genome = "hg20",
    n.cores = 2
  )

  return(out)
})

# save for future use
save(copykat.output, file = 'copyKatOutput.rdata')

# collect the copy number output
ls = list.files(pattern = '*_copykat_CNA_raw_results_gene_by_cell.txt')

# read each copy number result in
ls = lapply(ls, function(x) {
  read.table(x, header = T, row.names = NULL)
})
names(ls) = gsub('\\_.*',
                 '',
                 list.files(pattern = '*_copykat_CNA_raw_results_gene_by_cell.txt'))

# combine all the results by chromosome
```

```
ls = lapply(ls, function(x) {

  # aggregate together the output by chromosome
  out = aggregate(x[, 8:ncol(x)], by = list(x$chromosome_name), mean)
  rownames(out) = out[, 1]
  out = out[,-1]

  # record the mean values for each chromosome
  out = apply(out, 1, mean)
  return(out)

})

# combine all the results together
ls = do.call(cbind, ls)

# record values with chr14 deletion
fourteen = ls[14,] < 0
fourteen = as.factor(fourteen)
levels(fourteen) = c('Wildtype', 'Deleted')

# record the original samples into a new metadata characteristic called fourteen
mRcc$fourteen = mRcc$orig.ident

# record the names of the original samples
nams = unique(mRcc$fourteen)

# make sure the order of fourteen matches the order of nams
fourteen = fourteen[match(nams, names(fourteen))]

# make sure fourteen is a character not a factor
fourteen = as.character(fourteen)

# write function to sub in deleted or wildtype for each sample
subin <- function(vector, list){

  # for each element in the list, replace the element of the vector which matches
  # the named element of the list with the element of the list
  for (x in 1:length(list)){
    vector[vector == names(list)[x]] = list[[x]]
  }

  # return the vector
  return(vector)
}

# store fourteen
mRcc$fourteen = subin(mRcc$fourteen, fourteen)
```

8.4: Immune cell analysis

```r
# extract the immune compartment from the mRCC object
imm = subset(mRcc,
             sctype_classification %in% c('Unknown',
                                          'Immune cells',
                                          'Hematopoietic cells'))

# get the new variable features of the data
imm <- FindVariableFeatures(imm,
                            selection.method = "vst",
                            nfeatures = 2000,
                            assay = 'joined')

# run PCA
imm <-
  RunPCA(imm,
         features = VariableFeatures(object = imm, assay = 'joined'))

# make sure the default assay is joined
DefaultAssay(imm) = 'joined'

# make a heatmap to help decide on the clusters
DimHeatmap(imm,
           dims = 1:15,
           cells = 500,
           balanced = TRUE)

# make an elbow plot to help decide on number of clustering dimensions
ElbowPlot(imm)

# set the resolution range
resolution.range <- seq(from = 0, to = 1, by = 0.2)

imm <- imm %>%

  # run UMAP using harmoney
  RunUMAP(reduction = "harmony", dims = 1:6) %>%

  # find the nearest neighbors
  FindNeighbors(reduction = "harmony", dims = 1:6) %>%

  # find the clusters across the resolution range
  FindClusters(resolution = resolution.range) %>%
  identity()

# display clustree
clustree(imm, prefix = "joined_snn_res.")
# choose an ideal clustering based on probabilities

Idents(imm) = imm$joined_snn_res.0.4

# load gene set preparation function
```

```r
source(
  "https://raw.githubusercontent.com/IanevskiAleksandr/sc-type/master/R
  /gene_sets_prepare.R"
)
# load cell type annotation function
source(
  "https://raw.githubusercontent.com/IanevskiAleksandr/sc-type/master/R
  /sctype_score_.R"
)
# DB file
db_ <-
  "https://raw.githubusercontent.com/IanevskiAleksandr/sc-type/master
/ScTypeDB_full.xlsx"

tissue <- "Immune system"

# prepare gene sets
gs_list <- gene_sets_prepare(db_, tissue)

# check Seurat object version (scRNA-seq matrix extracted differently in Seurat v4/v5)
seurat_package_v5 <-
  isFALSE('counts' %in% names(attributes(imm[["joined"]])))

# extract scaled scRNA-seq matrix
scRNAseqData_scaled <-
  if (seurat_package_v5) {
    as.matrix(imm[["joined"]]$scale.data)
  } else {
    as.matrix(imm[["joined"]]@scale.data)
  }


# run ScType
es.max <-
  sctype_score(
    scRNAseqData = scRNAseqData_scaled,
    scaled = TRUE,
    gs = gs_list$gs_positive,
    gs2 = gs_list$gs_negative
  )

# merge by cluster
cL_resutls <-
  do.call("rbind", lapply(unique(imm@meta.data$joined_snn_res.0.4),
                        function(cl) {
    es.max.cl = sort(rowSums(es.max[, rownames(imm@meta.data[
      imm@meta.data$joined_snn_res.0.4 ==cl,
      ])]), decreasing = !0)
    head(data.frame(
      cluster = cl,
      type = names(es.max.cl),
      scores = es.max.cl,
      ncells = sum(imm@meta.data$joined_snn_res.0.4 == cl)
    ),
    10)
  }))

# collect the most likely cell type for each cluster
sctype_scores <-
  cL_resutls %>% group_by(cluster) %>% top_n(n = 1, wt = scores)
sctype_scores = as.data.frame(sctype_scores)

# set low-confident (low ScType score) clusters to "unknown"
sctype_scores$type[as.numeric(as.character(sctype_scores$scores)) <
                     as.numeric(as.character(sctype_scores$ncells)) /
                     4] <- "Unknown"

# add a new metadata slot which includes the sctype classification
imm@meta.data$sctype_classification = ""
for (j in unique(sctype_scores$cluster)) {
  cl_type = sctype_scores[sctype_scores$cluster == j, ]

  imm@meta.data$sctype_classification[imm@meta.data$joined_snn_res.0.4 == j] =
    as.character(cl_type$type[1])
}

# reset the Idents to match the new cell types
Idents(imm) = imm$sctype_classification

# save for future reference
save(imm, file = './immRCC.rdata')

# now find markers for each cell type between wildtype and deleted samples
markers = lapply(unique(imm$sctype_classification), function(x) {

  # extract the given cell type
  out = subset(imm, sctype_classification == x)

  # set Idents to fourteen
  Idents(out) = out$fourteen

  # find all markers with no filters
  marks <- FindAllMarkers(
    out,
    only.pos = F,
    return.thresh = 1,
    logfc.threshold = 0
  )
  return(marks)
})

# extract only comparisons with at least one differentially expressed gene
names(markers) = unique(imm$sctype_classification)
markers = markers[unlist(lapply(markers, function(x) nrow(x) > 0))]

# collect the genes of interset for graphing
```

```
genes_receptor = c('HAVCR2',
                   'TIGIT',
                   'LAG3',
                   'CTLA4',
                   'PDCD1',
                   'VSIR',
                   'CD47')

# get the markers which correspond the genes of interest for T cells
t.cells = markers[c(2,3,5)]

# extract only interesting genes
t.cells = lapply(t.cells, function(x) return(x[x$gene %in% genes_receptor,]))

# combine the T cells together
t.cells = do.call(rbind, t.cells)

# make the cell names more readable
t.cells$cell = gsub("\\..*", "", rownames(t.cells))

# add a column called group
t.cells$group = 'T cells'

# Figure 4E
tc = ggplot(t.cells, aes(
  x = gene,
  y = cell,
  size = -log(p_val_adj, 10),
  color = avg_log2FC
)) +
  geom_point() +
  ggh4x::facet_grid2(~ cluster, scales = 'free', independent = 'x') +
  theme_classic() +
  theme(text = element_text(size = 15),
        axis.text.x = element_text(
          angle = 90,
          vjust = 0.5,
          hjust = 1
        )) +
  scale_colour_gradient2(
    low = "blue",
    mid = "gray",
    high = "red",
    midpoint = 0,
    space = "Lab",
    na.value = "grey50",
    guide = "colourbar",
    aesthetics = "colour"
  ) +
  xlab('') +
  ylab('')

saveRDS(tc, '<figure 4e>')
```

```r
# extract the T cell compartment of the imm object
t.cell = subset(imm, sctype_classification == 'Naive CD8+ T cells')

# set the Ident to fourteen
Idents(t.cell) = t.cell$fourteen

# Figure 4D
gridExtra::grid.arrange(
  VlnPlot(t.cell, features = c('KLRG1'), pt.size = 0) +
    stat_compare_means(label = 'p.signif', label.x = 1.5) +
    scale_y_continuous(expand = c(0, .2)) +
    theme(
      legend.position = 'none',
      axis.text.x = element_text(angle = 0, hjust = .5)
    ) +
    xlab('') +
    scale_fill_manual(values = c('black', 'purple')),
  VlnPlot(t.cell, features = c('IL7R'), pt.size = 0) +
    stat_compare_means(label = 'p.signif', label.x = 1.5) +
    scale_y_continuous(expand = c(0, .2)) +
    theme(
      legend.position = 'none',
      axis.text.x = element_text(angle = 0, hjust = .5)
    ) +
    xlab('') +
    scale_fill_manual(values = c('black', 'purple')),
  VlnPlot(t.cell, features = c('IFNG'), pt.size = 0) +
    stat_compare_means(label = 'p.signif', label.x = 1.5) +
    scale_y_continuous(expand = c(0, .2)) +
    theme(
      legend.position = 'none',
      axis.text.x = element_text(angle = 0, hjust = .5)
    ) +
    xlab('') +
    scale_fill_manual(values = c('black', 'purple')),
  nrow = 1
)
```