The Journal of Open Source Software

gesel: a JavaScript package for client-side gene set enrichment

Aaron T. L. Lun [1] **and Jayaram Kancherla** [1]

1 Genentech Inc., South San Francisco, United States of America

DOI: 10.21105/joss.05777

Software

- Review
- Repository
- Archive

Editor: Arfon Smith

Reviewers:

- @majensen
- @bede

Submitted: 15 March 2023
Published: 24 October 2023

Summary

`gesel` is a JavaScript package for performing gene set enrichment analyses within the browser. All calculations are performed on the client device, without any no need for a dedicated backend server. This eliminates concerns around cost, scalability, latency, and data ownership that are associated with a backend-based architecture. We demonstrate the use of `gesel` with a basic web application that performs enrichment analyses on user-supplied genes with sets derived from the Gene Ontology and MSigDB. Developers can also use `gesel` to incorporate gene set enrichment capabilities into their own applications.

Statement of need

Gene set enrichment analyses (GSEA) are commonly used to interpret the biological activity of a user-supplied list of interesting genes (Subramanian et al., 2005). Briefly, this task involves quantifying the enrichment of each reference gene set's members inside the user-supplied list, where the reference sets are derived from a variety of sources such as previous experimental studies or *de novo* computational analyses. GSEA allows scientists to summarize a large list of gene identifiers into a tangible biological concept such as "syntaxin binding" or "T cell receptor signaling pathway". User-supplied lists are typically derived from differential expression analyses of transcriptome-wide assays like RNA sequencing, but any list of genes can be used, e.g., cluster-specific marker lists from single-cell RNA sequencing studies.

Given the popularity of GSEA in transcriptomics, it is not surprising that many software tools are already available to perform this analysis. Most existing GSEA tools operate inside frameworks like R/Bioconductor (Korotkevich et al., 2021; Wu & Smyth, 2012; Young et al., 2010) and require both installation of software and associated programming knowledge to use. Web applications like Enrichr and GeneTrail (Backes et al., 2007; Chen et al., 2013) provide more user-friendly interfaces that require minimal computational knowledge, targeted to the majority of bench scientists. These applications use a conventional backend architecture where the browser sends a request containing the user-supplied list of genes to a backend server; the backend then performs the analysis and returns the results to the user's device (i.e., the client) for inspection.

While common, this backend-based architecture is subject to a number of concerns around cost, scalability, latency, and data ownership. The application maintainer is responsible for provisioning, deploying, monitoring and maintaining a backend server, which requires both money and time. The maintainer is also responsible for scaling up the backend compute in response to increased usage, further increasing costs in an unpredictable manner. The user-supplied lists need to be transferred to the backend and the results need to be transferred back to the client, introducing latency to the user experience. Finally, the fact that the user's inputs are accessible to the backend introduces potential issues of data ownership, e.g., for confidential biomarker lists or signatures.

Here, we present gesel (https://npmjs.com/package/gesel), a JavaScript library for gene set enrichment analyses that operates fully inside the client. Web applications can easily incorporate gesel via the standard npm installation process, enabling developers to create user-friendly interfaces for GSEA in different contexts. The browser will then handle all GSEA-related computation within these applications, eliminating the responsibility of maintaining a backend and avoiding any transfer of user data. This obviates the problems associated with a backend architecture and allows the application to scale naturally to any number of user devices. We demonstrate the use of gesel by creating a simple web application (https://ltla.github.io/gesel-app) for identifying interesting gene sets based on overlaps with user-supplied lists.

Usage

gesel's analysis involves testing for significant overlap between each reference gene set and the user-supplied list of genes. While this is the simplest form of GSEA, it is fast, intuitive, mostly effective and avoids the need for users to specify a ranking across the supplied genes. The algorithm can also be phrased as a search for the gene sets that contain at least one entry of the user-supplied list. To demonstrate, consider the following list of gene symbols mixed with Ensembl and Entrez identifiers.

```
let user_supplied = [ "SNAP25", "NEUROD6", "ENSG00000123307", "1122" ];
```

Our first task is to map these user-supplied gene identifiers to gesel's internal identifiers. In this case, we are interested in human gene sets, hence the taxonomy identifier in the searchGenes() call.

```
let input_mapped = await gesel.searchGenes("9606", user_supplied);
console.log(input_mapped);
// [ [ 4639 ], [ 12767 ], [ 12577 ], [ 828 ] ]
```

To simplify matters, we will ony use the first matching gesel gene identifier for each user-supplied gene. Other applications may prefer to handle multi-mapping genes by, e.g., throwing an error to require clarification from the user.

```
let input_list = [];
for (const x of input_mapped) {
    if (x.length >= 1) {
        input_list.push(x[0]);
    }
}
console.log(input_list);
// [ 4639, 12767, 12577, 828 ]
```

We call findOverlappingSets() to search for all human gene sets that overlap the user-supplied list. This returns an array of objects with the set identifier, the number of overlapping genes, the size of each set and the enrichment p-value based on the hypergeometric distribution. Applications can sort this array by the p-value to prioritize sets with significant overlap.

```
let overlaps = await gesel.findOverlappingSets("9606", input_list);
console.log(overlaps.length);
// 935

console.log(overlaps[0]);
// { id: 379, count: 1, size: 10, pvalue: 0.0009525509051785397 }
```

Given a set identifier, we obtain that set's details with the fetchSingleSet() function.

```
let set_details = await gesel.fetchSingleSet("9606", overlaps[0].id);
console.log(set_details);
```

```
// {
//   name: 'GO:0001504',
//   description: 'neurotransmitter uptake',
//   size: 10,
//   collection: 0,
//   number: 379
// }
```

The same approach can also be used to obtain the details of the collection containing that set.

```
let parent_collection = await gesel.fetchSingleCollection("9606", set_details.collection
console.log(parent_collection);
// {
//   title: 'Gene ontology',
//   description: 'Gene sets defined from the Gene Ontology (version 2022-07-01), source
//   species: '9606',
//   maintainer: 'Aaron Lun',
//   source: 'https://github.com/LTLA/gesel-feedstock/blob/gene-ontology-v1.0.0/go/build
//   start: 0,
//   size: 18933
// }
```

The membership of each set is obtained with the `fetchGenesForSet()` function. This returns an array of `gesel`'s internal gene identifiers, which can be mapped to various standard identifiers or symbols using the `fetchAllGenes()` function.

```
let set_members = await gesel.fetchGenesForSet("9606", overlaps[0].id);
console.log(set_members);
// Uint32Array(10) [
//     343, 1452, 2222,
//    4543, 4547, 4548,
//    4639, 6238, 6246,
//   14046
// ]

let all_symbols = (await gesel.fetchAllGenes("9606")).get("symbol");
console.log(Array.from(set_members).map(i => all_symbols[i]));
// [
//   [ 'ATP1A2' ],
//   [ 'SLC29A1' ],
//   [ 'SLC29A2' ],
//   [ 'SLC1A3' ],
//   [ 'SLC1A6' ],
//   [ 'SLC1A7' ],
//   [ 'SNAP25' ],
//   [ 'SYNGR3' ],
//   [ 'SLC6A5' ],
//   [ 'SLC38A1' ]
// ]
```

Each set also has some associated free text in its name and description. `gesel` can query this text to find sets of interest, with some basic support for the `?` and `*` wildcards.

```
let hits = await gesel.searchSetText("9606", "B immunity");
let first_hit = await gesel.fetchSingleSet("9606", hits[0]);
// {
//   name: 'GO:0019724',
//   description: 'B cell mediated immunity',
//   size: 4,
//   collection: 0,
//   number: 5715
// }

let hits2 = await gesel.searchSetText("9606", "B immun*");
let first_hit2 = await gesel.fetchSingleSet("9606", hits2[0]);
// {
//   name: 'GO:0002312',
//   description: 'B cell activation involved in immune response',
//   size: 2,
//   collection: 0,
//   number: 858
// }
```

The output of `searchSetText()` can then be combined with the output of `findOverlappingSets()` to implement advanced searches in downstream applications.

To demonstrate `gesel`'s functionality, we developed a simple web application that tests for gene set enrichment among user-supplied genes (A. Lun & Kancherla, 2023). Given several parameters such as a list of user-supplied genes and a free-text query, the application shows a table containing the gene sets that satisfy the search parameters. Sets are sorted by increasing p-value to focus on those with significant enrichment. Clicking on a row corresponding to a particular gene set shows the identities of its genes, with emphasis applied to those in the user-supplied list. The parameters of each search are captured by query strings, allowing users to easily save and share searches by copying the URL from the browser's address bar. More adventurous users can also navigate a 2-dimensional embedding (Van der Maaten & Hinton, 2008) of gene sets, where sets with similar members are placed next to each other on the embedding; this provides an alternative representation of the search results that encourages exploration of related gene sets.

Implementation details

`gesel` supports two modes of operation - a "full client-side" mode and a more lightweight "on-demand" mode. These differ with respect to how they obtain the database files containing the reference gene sets. In full client-side mode, `gesel` will download the relevant database files from the static file server to the client. All calls to `gesel` functions will then perform queries directly on the downloaded files. In this mode, the user pays an up-front cost for the initial download such that all subsequent calculations are fully handled within the client. This avoids any further network activity and the associated latency. For many applications, the up-front cost is likely to be modest - for example, the total size of the default human gene set database is just over 9 MB - so full client-side operation is simple and practical in most cases.

In the on-demand mode, `gesel` will perform HTTP range requests to fetch relevant slices of each database file. For example, `findOverlappingSets()` needs to obtain the mapping of each gene to the gene sets of which it is a member. Rather than downloading the entire mapping file, `gesel` will ask the server to return the range of bytes containing only the mapping for the desired gene. This is inspired by similar strategies for querying genomics data (Kancherla et al., 2020) and reduces the burden on the client device and network. Range requests are suited for applications that expect only sporadic usage of `gesel` such that an up-front download of the entire database cannot be justified. They are also more scalable as the number of gene sets increases into the millions, where an up-front download may become too large to be practical. Obviously, using this mode involves increased network activity and latency from multiple range requests if `gesel` functions are frequently called. This is partially mitigated by `gesel`'s transparent caching of responses in memory.

In both cases, we stress that `gesel` only requires a static file server to host the database files and optionally to support range requests. We do not have to provision and maintain a dedicated back-end server to handle the `gesel` queries, saving time and money; rather, any generic static server can be used, including free offerings, e.g., from GitHub. The client machine performs all of the calculations and the user receives the results immediately on completion, enabling low-latency applications that minimize network traffic. Similarly, there is no transfer of user-supplied gene lists to an external server, avoiding any questions over data ownership. Most importantly, as each user brings their own compute to the application, it scales to any number of users at no cost to us (i.e., the `gesel` maintainers). Indeed, we consider `gesel`'s development to be a natural consequence of the "client-side compute" philosophy described in A. T. L. Lun & Kancherla (2023).

`gesel` works with any database files prepared according to the contract outlined in the feedstock repository (A. Lun, 2023). These are simple tab-separated text files containing information about the genes, sets, collections, and the mappings between them. We store the byte ranges for each relationship in the mapping files to enable on-demand range requests. To reduce data transfer, we apply some standard practices like delta-encoding the sorted gene identifiers and Gzip-compressing the byte range files. `gesel`'s default database incorporates public gene sets from the Gene Ontology (Ashburner et al., 2000) and, for human and mouse, the majority of the relevant MSigDB subcollections (Liberzon et al., 2011). However, application developers can easily point `gesel` to a different database by overriding the request URL. For example, we adapted the scripts in the feedstock repository to create a company-specific database of custom gene sets based on biomarker lists and other signatures. This is hosted inside our internal network for use by our in-house `gesel`-based applications.

Acknowledgements

Thanks to Chris Bolen, Alejandro Chibly, Brandon Kayser and Xiangnan Guan, for the scientific questions that motivated the development of this library; Hector Corrada Bravo, for his feedback on the uselessness of the early versions of the free-text search; and Allison Vuong and Luke Hoberecht, for recovering ATLL's scarf when he forgot it while thinking about the library design during a team dinner.

References

Ashburner, M., Ball, C. A., Blake, J. A., Botstein, D., Butler, H., Cherry, J. M., Davis, A. P., Dolinski, K., Dwight, S. S., Eppig, J. T., Harris, M. A., Hill, D. P., Issel-Tarver, L., Kasarskis, A., Lewis, S., Matese, J. C., Richardson, J. E., Ringwald, M., Rubin, G. M., & Sherlock, G. (2000). Gene ontology: tool for the unification of biology. The Gene Ontology Consortium. *Nat Genet*, *25*(1), 25–29. https://doi.org/10.1038/75556

Backes, C., Keller, A., Kuentzer, J., Kneissl, B., Comtesse, N., Elnakady, Y. A., Iler, R., Meese, E., & Lenhof, H. P. (2007). GeneTrail–advanced gene set enrichment analysis. *Nucleic Acids Res*, *35*(Web Server issue), W186–192. https://doi.org/10.1093/nar/gkm323

Chen, E. Y., Tan, C. M., Kou, Y., Duan, Q., Wang, Z., Meirelles, G. V., Clark, N. R., & Ma'ayan, A. (2013). Enrichr: interactive and collaborative HTML5 gene list enrichment analysis tool. *BMC Bioinformatics*, *14*, 128. https://doi.org/10.1186/1471-2105-14-128

Kancherla, J., Yang, Y., Chae, H., & Corrada Bravo, H. (2020). Epiviz File Server: Query, transform and interactively explore data from indexed genomic files. *Bioinformatics*, *36*(18), 4682–4690. https://doi.org/10.1093/bioinformatics/btaa591

Korotkevich, G., Sukhov, V., Budin, N., Shpak, B., Artyomov, M. N., & Sergushichev, A. (2021). Fast gene set enrichment analysis. *bioRxiv*. https://doi.org/10.1101/060012

Liberzon, A., Subramanian, A., Pinchback, R., Thorvaldsdóttir, H., Tamayo, P., & Mesirov, J. P. (2011). Molecular signatures database (MSigDB) 3.0. *Bioinformatics*, *27*(12), 1739–1740. https://doi.org/10.1093/bioinformatics/btr260

Lun, A. (2023). *Build gene sets to feed gesel*. https://github.com/LTLA/gesel-feedstock

Lun, A. T. L., & Kancherla, J. (2023). Powering single-cell analyses in the browser with WebAssembly. *Journal of Open Source Software*, *8*(89), 5603. https://doi.org/10.21105/joss.05603

Lun, A., & Kancherla, J. (2023). *Gesel gene set search*. https://ltla.github.io/gesel-app

Subramanian, A., Tamayo, P., Mootha, V. K., Mukherjee, S., Ebert, B. L., Gillette, M. A., Paulovich, A., Pomeroy, S. L., Golub, T. R., Lander, E. S., & Mesirov, J. P. (2005). Gene set enrichment analysis: a knowledge-based approach for interpreting genome-wide expression profiles. *Proc Natl Acad Sci U S A*, *102*(43), 15545–15550. https://doi.org/10.1073/pnas.0506580102

Van der Maaten, L., & Hinton, G. (2008). Visualizing data using t-SNE. *Journal of Machine Learning Research*, *9*(86), 2579–2605. http://jmlr.org/papers/v9/vandermaaten08a.html

Wu, D., & Smyth, G. K. (2012). Camera: a competitive gene set test accounting for inter-gene correlation. *Nucleic Acids Res*, *40*(17), e133. https://doi.org/10.1093/nar/gks461

Young, M. D., Wakefield, M. J., Smyth, G. K., & Oshlack, A. (2010). Gene ontology analysis for RNA-seq: accounting for selection bias. *Genome Biol*, *11*(2), R14. https://doi.org/10.1186/gb-2010-11-2-r14