Supplementary materials

data_downloader module

Data_downloader is responsible for automatically collecting data from online resources. First, we wrote a script to collect sigma factor regulations, transcriptional factor regulations, and sRNA regulations from RegulonDB and this script will be invoked every week to check whether the source database is updated. We also wrote other scripts for collecting data from the other resources. Then experimental protein-protein interactions (PPI) were obtained from the STRING database. In the case of chemical–enzyme interaction (CPI), a small set of CPI (e.g., activating compound and inhibitor compound) were collected from the BRENDA database. Chemical-TF interactions from RegulonDB were then added. To add more chemical-protein interactions, we extracted the interaction data of E.coli from the STITCH database. The genome-scale metabolic model of E. coli iML1515 was downloaded from BiGG. The annotation of metabolites, reactions, and pathways was derived from the model, while the annotation of genes was derived from NCBI.

data_filter module

Data_filter module filters data from different data sources using different processing flows. For example, gene-reaction, reaction–metabolite, and pathway-reaction relationships were obtained by directly parsing the metabolic network model as well as the annotation of metabolites, reactions, and pathways. In the case of the reaction–metabolite links, the links through currency metabolites (1) were removed to obtain biologically meaningful pathways using the new ranked pair method (2). CPI data obtained from the BRENDA database were filtered using the metabolites in the model. For the PPI data obtained from the STRING database, we used two criteria to filter: 1) marked as ‘experimental’; 2) the experimental score was greater than 700. However, for CPI data from STITCH, in addition to the same rules of PPI, other rules need to be introduced: 1) using the STITCH alias file to match the STITCH chemical ID with metabolite name; 2) interaction types should be ‘binding’, ‘activation’, and ‘inhibition’. 3) filtering the data using the metabolites in the metabolic network. The data processing of RegulonDB is simpler, involving only the matching of gene IDs and metabolite IDs, but it should be noted that this database is specialized for E. coli (other species do not necessarily have
such high-quality database), involving sigma factor regulations, transcriptional factor regulations, sRNA regulations, etc.

Data integrator module

Data integrator module was used to convert the filtered data to tabular files. In general, the filtered data represented as five classes of interactions: (1) regulation of a gene/protein by a metabolite (e.g., CPI); (2) regulatory interactions between genes/proteins (e.g., TFGI and PPI); (3) interactions between genes and reactions (GRIs); (4) interactions between metabolites and reactions (MRIs); (5) interactions between pathways and reactions (PRIs). We integrated these distributed interaction data from various sources based on shared metabolites, genes and reactions, and export this integrated data into Gremlin load data format (including column headers such as ~id, ~from, ~to, ~label, attribute, etc.) for easy writing to AWS Neptune via Gremlin.

The graph database schema

Using a graph database, it's easy to model for highly connected data and provides simpler and more efficient ways to perform complex queries. In this study, we adopt AWS Neptune, a fast, reliable, and fully-managed graph database. In this case, we no longer need to worry about database management tasks like hardware provisioning, software patching, setup, configuration, or backups. Neptune supports the popular graph query languages Apache TinkerPop Gremlin and W3C's SPARQL, so we can efficiently query relationships between nodes with milliseconds latency.

The graph database schema consists of nodes, edges, and properties. A node represents an entity, which may have labels and properties, one of which is a key property that enables its unique identification. In ERMer, there are four types (Gene, Metabolite, Reaction, and Pathway) of entities. For simplicity, protein is not used as an entity and is represented by its corresponding gene. There are nine types (RPI, RMI, MRI, GRI, TFGI, CPI, SFGI, sRGI, and PPI) of edges classified based on biological function in ERMer. It should be noted that TFGI, SFGI, sRGI, and PPI are all represented as edges between genes in the graph database. Finally, ERMer currently comprises 8421 nodes and 36331 edges.

The cloud platform implementation
The cloud platform development adopts the frontend and backend separation techniques. For the serverless backend, we mainly use AWS Lambda and AWS Neptune to build a backend process for handling requests from the frontend. AWS Neptune was used as the database backend to store ERMer nodes and edges. The nodes and edges files are in the gremlin load_data format and stored on the Amazon S3 bucket. When a user sends a request from the client, the request will be forwarded to the AWS Lambda function through API Gateway. In our case, the Lambda function has three main functions to handle the different types of query requests and invoke the corresponding gremlin API to query the requested data from the graph database.

Natively, AWS Neptune supports graph visualization based on Jupyter Notebook. However, it requires specialized knowledge in query languages such as gremlin and a deep understanding of the underlying structure of graphs, which is not easy for the end-users. In addition, it can not be used interactively. For this purpose, we have integrated the AWS Neptune graph database with the serverless AWS lambda function and frontend G6 graph visualization engine, facilitating end-users to search, visualize and navigate our graph database without the need to write any querying program. In our platform, after the results in gremlin output format are obtained, the Lambda function will extract nodes and edges, and invoke another round of gremlin queries to get the corresponding attributes for nodes and edges. Then all the information, including nodes, edges, and attributes, can be presented in a graph with the G6 graph visualization engine. We use AWS S3 to host static web resources, including HTML, CSS, and JavaScript.

## Q&A Search

Q&A is implemented to retrieve key metabolites and regulatory factors/patterns. Several general questions are predefined for the convivence of users. For example, if the question “What are the key TFs regulating genes in both pathways?” is selected, all key TFs will be displayed in both table and graph manner after selecting the corresponding two pathways. In addition to TFs, we provide searches for important sRNAs, chemicals, and Sigma factors affecting both pathways. For example, pyridoxal 5′-phosphate, glycerol, and citrate are more involved in the regulation of ‘Glycine and Serine Metabolism’ and ‘Citric Acid Cycle’ pathways (Figure S4). For these questions, we use two ways to show the results. In the figure, we
highlight the regulatory factors to better show the regulatory relevance of the two pathways (Figure S4). In the table, we sort the regulators by the number of regulated genes in the pathway. The number of regulated genes and regulation patterns for each chemical will be counted separately for each pathway after clicking ‘Visualization’ (Figure S5). Besides, the linkage metabolites between the two specific pathways can be obtained by selecting the question “What are the key metabolites shared by both pathways?” (Figure S6). Then the top 10 of TFs, sRNAs, and chemicals with the most regulatory targets in *E. coli* can be queried in the Q&A, and statistical analysis is performed to demonstrate the distribution characteristics of regulatory factors in the pathway (Figure S8). ERMer also provides a hierarchical map of TFs in the Q&A search as well. Overall, the *E. coli* TF-TF network has an essentially feedforward layered structure, where feedback is mainly limited to autoregulation (3,4) (Figure S9).

```
with RECURSIVE le (idfrom,ldto,edgelabel,edgeattribute,path,depath,CYCLE) AS
(
  SELECT idfrom,ldto,edgelabel,edgeattribute,ARRAY[ldto] AS path,1 AS depath,false FROM ermer_edge WHERE idfrom="gly"
  UNION ALL
  SELECT e2.idfrom,e2.ldto,e2.edgelabel,e2.edgeattribute,e3.path+[e3.ldto,e3.depath+1,e2.idfrom=ANY(e3.path)]
  FROM ermer_edge e2,le e3 WHERE e3.idto=e2.idfrom AND NOT CYCLE
)
SELECT * from le WHERE le.idto="b2985"
```

Figure S1. PostgreSQL query statements.
Figure S2. Regulatory cascades filtering and redrawing.
Figure S3. The regulatory pattern of Crp.

Figure S4. Retrieval of the key metabolites regulating both pathways.
Figure S5. Retrieval of key chemicals regulating genes in both pathways. The results are shown in Table format.
Figure S6. Shared metabolites between 'Glycine and Serine Metabolism' and 'Citric Acid Cycle' pathways.
Figure S7. Retrieval of key TFs regulating genes across pathways.
Figure S8. Top 10 TFs involved in the TF-Gene regulations.
Figure S9. The hierarchical structure of the TF-TF network in *E. coli*.

Reference

1. Ma, H. and Zeng, A.-P. (2003) Reconstruction of metabolic networks from genome data and analysis of their global structure for various organisms. *Bioinformatics*, 19, 270-277.

2. Gao, Y., Yuan, Q., Mao, Z., Liu, H. and Ma, H. (2021) Global connectivity in genome-scale metabolic networks revealed by comprehensive FBA-based pathway analysis. *BMC Microbiology*, 21, 292.

3. Ma, H.-W., Kumar, B., Ditges, U., Gunzer, F., Buer, J. and Zeng, A.-P. (2004) An extended transcriptional regulatory network of *Escherichia coli* and analysis of its hierarchical structure and network motifs. *Nucleic Acids Res.*, 32, 6643-6649.

4. Ma, H.-W., Buer, J. and Zeng, A.-P. (2004) Hierarchical structure and modules in the *Escherichia coli* transcriptional regulatory network revealed by a new top-down approach. *BMC Bioinformatics*, 5, 199.