PloverDB: A high-performance platform for serving biomedical knowledge graphs as standards-compliant web APIs

A PREPRINT

Amy K. Glen[1,*], Eric W. Deutsch[2], and Stephen A. Ramsey[1,3]

[1]School of Electrical Engineering and Computer Science, Oregon State University, Corvallis, Oregon, USA
[2]Institute for Systems Biology, Seattle, Washington, USA
[3]Department of Biomedical Sciences, Oregon State University, Corvallis, Oregon, USA
[*]To whom correspondence should be addressed

March 8, 2025

ABSTRACT

Knowledge graphs are increasingly being used to integrate heterogeneous biomedical knowledge and data. General-purpose graph database management systems such as Neo4j are often used to host and search knowledge graphs, but such tools come with overhead and leave biomedical-specific standards compliance and reasoning to the user. Interoperability across biomedical knowledge bases and reasoning systems necessitates the use of standards such as those adopted by the Biomedical Data Translator consortium. We present PloverDB, a comprehensive software platform for hosting and efficiently serving biomedical knowledge graphs as standards-compliant web application programming interfaces. In addition to fundamental back-end knowledge reasoning tasks, PloverDB automatically handles entity resolution, exposure of standardized metadata and test data, and multiplexing of knowledge graphs, all in a single platform designed specifically for efficient query answering and ease of deployment. PloverDB increases data accessibility and utility by allowing data providers to quickly serve their biomedical knowledge graphs as standards-compliant web services.
Availability and Implementation: PloverDB's source code and technical documentation are publicly available under an MIT License at github.com/RTXteam/PloverDB.
Contact: amykglen@gmail.com

1 Introduction

Knowledge graphs (KGs) have emerged as useful abstractions for integrating heterogeneous biomedical data, making them convenient substrates for data mining and other computational reasoning techniques. The way in which such knowledge graphs are exposed for use varies; some graphs are only available for flat-file download, while others are exposed via a web application programming interface (API) or user interface, often backed by a general-purpose graph database management system such as Neo4j (github:neo4j/neo4j). Naturally, a knowledge graph's ultimate utility is heavily influenced by the convenience, speed, and reliability of its mode of access.

The National Center for Advancing Translational Sciences (NCATS) Biomedical Data Translator [2] (abbreviated as "Translator") is a distributed knowledge graph-based computing system that integrates and reasons across disparate biomedical data to answer questions like *"What drugs might be repurposed to treat Fanconi anemia?"*. The Translator Consortium has defined data standards for biomedical knowledge graphs, including 1) the *Biolink Model*, a standard schema and semantic layer for biomedical KGs [7], and 2) *TRAPI*, a standard JavaScript Object Notation (JSON) web API format for biomedical KGs (github:NCATSTranslator/ReasonerAPI).

Within Translator, knowledge graphs are exposed via TRAPI-compliant web APIs that can answer single-edge or "one-hop" pattern-matching queries, such as *Acetaminophen—interacts_with→Protein?* (see Sec. 2.1.1), and in

doing so, perform certain fundamental reasoning tasks like transitive chaining of concept subclass relationships (see Sec. 2.1.2). Such KG APIs can then be used by reasoning systems, such as Translator reasoning agents (e.g., [3], [1], github:ranking-agent/aragorn) or large language model-based chain-of-thought reasoning systems, to answer larger queries.

As Translator's requirements for interoperability and graph-based reasoning illustrate, transitioning from a flat-file version of a knowledge graph to a deployed, standards-compliant web API is not a trivial task. The KG owner must 1) canonicalize their graph (i.e., identify and unify semantically identical concept nodes); 2) import it into some sort of database platform (often Neo4j); 3) convert incoming TRAPI queries into queries runnable on the back-end database system; 4) encode various semantic and other fundamental reasoning tasks such as transitive chaining of subclass relationships; 5) ensure that answers are transformed back into TRAPI format; 6) compute a meta knowledge graph and test triples; and 7) deploy the entire service as a performant web API, even in the face of heavy concurrent load.

Previous work on web-accessible hosting frameworks for biomedical knowledge graphs includes Plater, a tool for exposing a Neo4j-hosted Biolink-compliant KG as a TRAPI web API, which can be combined with Babel and ORION to achieve entity resolution and KG formation/import, respectively.[1] Another tool, BioThings Software Development Kit [5], can dynamically create a Biolink-compliant API for a KG according to a custom data parser, which can then be exposed as a TRAPI API by BioThings Explorer [1].

Despite previous efforts, there is a need for a platform that allows data owners to easily deploy Biolink-compliant flat-file knowledge graphs as TRAPI web APIs that 1) is *comprehensive* in terms of back-end reasoning (e.g., entity resolution, concept subclass chaining) and provided artifacts/endpoints (e.g., meta KG, test triples), 2) is *highly-performant*, and 3) can be *independently* deployed with *minimal effort* upon KG content or knowledge representation changes. To fill this gap, we created PloverDB, an all-in-one platform for efficiently hosting and serving Biolink-compliant KGs as TRAPI web APIs. We describe PloverDB's implementation and usage in the following sections.

2 Implementation

PloverDB is a fully in-memory Python-based platform for hosting and serving Biolink-compliant knowledge graphs as TRAPI web APIs, designed specifically for improved query speed and ease of deployment. Its design prioritizes ease of use and minimal configuration, enabling rapid deployment of KGs with minimal overhead. PloverDB was initially created in 2021 to replace Neo4j as the back-end database platform for the RTX-KG2 knowledge graph [8], but was generalized in 2024 to 1) act as a standalone, fully TRAPI-compliant web service and 2) be compatible with any Biolink-compliant knowledge graph.

PloverDB is Dockerized for convenient deployment, as depicted in Figure 1. During its Docker image build, PloverDB 1) downloads the Biolink-compliant KG nodes and edges files—which may be in either TSV or JSON Lines Knowledge Graph Exchange (KGX) format (github.com/biolink/kgx)—from user-specified public URLs provided in a JSON configuration file; 2) loads the graph into memory; 3) builds its core data structure and other indexes, utilizing the Biolink Model and Translator Node Normalizer[2] service as applicable; 4) optionally canonicalizes the graph; and 5) saves indexes to disk within the image. When a Docker container is then run from that image, PloverDB loads its core data structure and indexes into memory and starts a uWSGI/Flask server that exposes the graph via a TRAPI-compliant API, providing various endpoints (only standardized endpoints are depicted in Figure 1).

PloverDB's core data structure is essentially a nested adjacency dictionary in which nodes are mapped to their neighbors, organized by the categories of those neighbors and the types of edges that connect them to those neighbors. PloverDB does not use any intermediary database platforms such as Neo4j or SQLite, and instead stores the graph in Python data structures (dictionaries and sets). At runtime, multiple uWSGI workers (processes) share these read-only data structures to handle concurrent requests.

PloverDB is configured for a given knowledge graph via a single JSON config file, where users provide public URLs from which their KG files should be downloaded, can specify how node/edge properties should be represented in TRAPI attributes, and can control other options like graph canonicalization (see Sec. 3).

2.1 Capabilities

PloverDB is designed to be a comprehensive platform for transforming a Biolink-compliant knowledge graph into a fully-functioning TRAPI-compliant API. Its capabilities in terms of supported queries (Sec. 2.1.1), built-in reasoning (Sec. 2.1.2), ease of deployment (Sec. 2.1.3), and performance (Sec. 2.1.4) are detailed below.

[1] github:TranslatorSRI/Plater, github:TranslatorSRI/Babel, and github:RobokopU24/ORION.

[2] github:TranslatorSRI/NodeNormalization

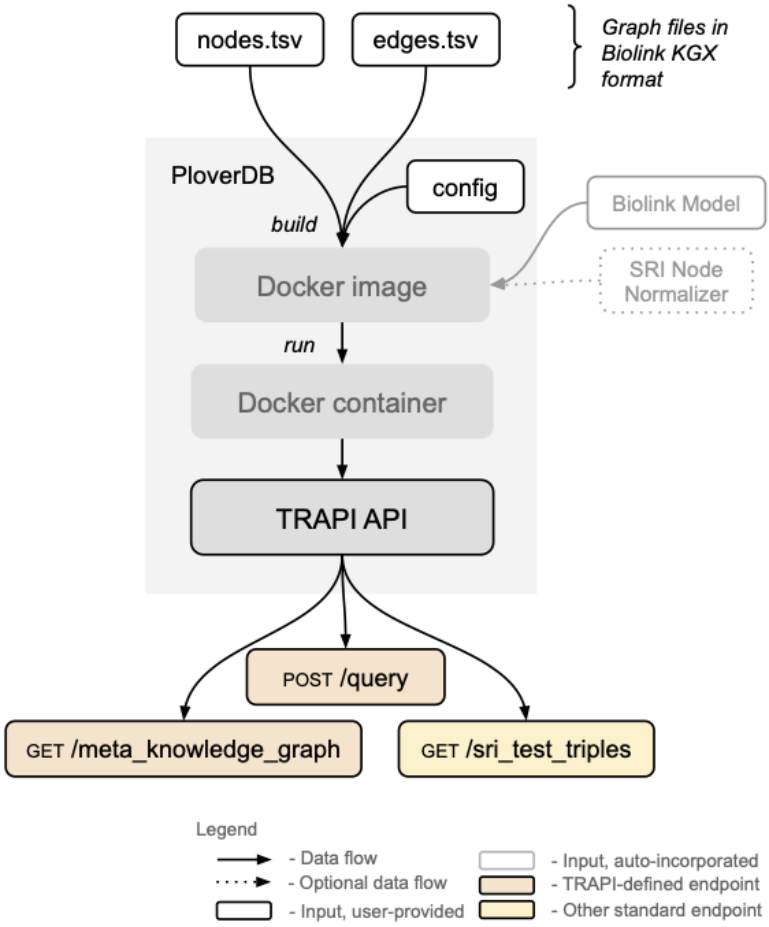

Figure 1: PloverDB's high-level architecture, depicting how an input KGX-formatted knowledge graph is exposed as a TRAPI web API. Details are described in the main text.

2.1.1 Query structure

PloverDB is designed to answer one-hop TRAPI queries, which are essentially graph-based pattern matching queries consisting of two nodes connected by a single edge[3]. In such a "query graph", nodes can be constrained by their concept (i.e., identifier) and semantic type (i.e., category) and edges can be constrained by their semantic type (i.e., predicate) and attributes (e.g., supporting publications, knowledge source, source type, etc.) Examples of such query graphs are provided in Appendix A.

2.1.2 Built-in reasoning

Importantly, when PloverDB answers the queries defined in Section 2.1.1, it returns all knowledge subgraphs that fulfill the query graph not only *directly*, but also according to:

- the *hierarchies* of node and edge types according to Biolink,
- the *symmetry* of edge types according to Biolink,
- the *canonicity* of edge types according to Biolink,
- concept *equivalency*, and
- concept *subclass relationships* (via transitive chaining) present in the underlying graph or in a user-specified external source.

[3]Reasoning systems answer more complex queries by stitching together one-hop KG queries.

Of note, concept equivalency is determined either by 1) identifiers provided in an `equivalent_identifiers` (or similar) property on each node in the KG, or, in the absence of such a property, 2) the Translator Node Normalizer service. PloverDB fulfills most of these reasoning tasks at build time and encodes them into its indexes for constant-time lookup at query time.

2.1.3 Simplified deployment

PloverDB is designed to streamline deployment so that data owners can easily and independently update their service upon KG content/representation changes. Thus, it:

1. is fully Dockerized and uses a standard Debian Linux-based base image[4] from the public DockerHub registry,
2. automatically constructs and exposes a TRAPI meta knowledge graph and corresponding test triples,
3. allows multiple KGs to be served from the same PloverDB application, accessible at separate sub-endpoints (i.e., "multiplexing" of KGs),
4. provides a built-in (authenticated) remote deployment mechanism, enabling data owners to remotely redeploy their service with minimal downtime,
5. provides endpoints to aid remote debugging, that expose log and KG/code version information, and
6. can (optionally) be registered in SmartAPI [9] using the TRAPI registration template[5].

2.1.4 Performance

PloverDB uses a fully in-memory architecture for efficient query answering. Based on a controlled study (methods detailed in Appendix B) of the performance of PloverDB in hosting the RTX-KG2 knowledge graph, PloverDB's response time (Fig. 2) appears asymptotically linear in terms of the number of edges in the query answer, taking about 0.01 seconds per 100 answer edges. PloverDB's speed and low overhead is a trade-off; its in-memory design means that

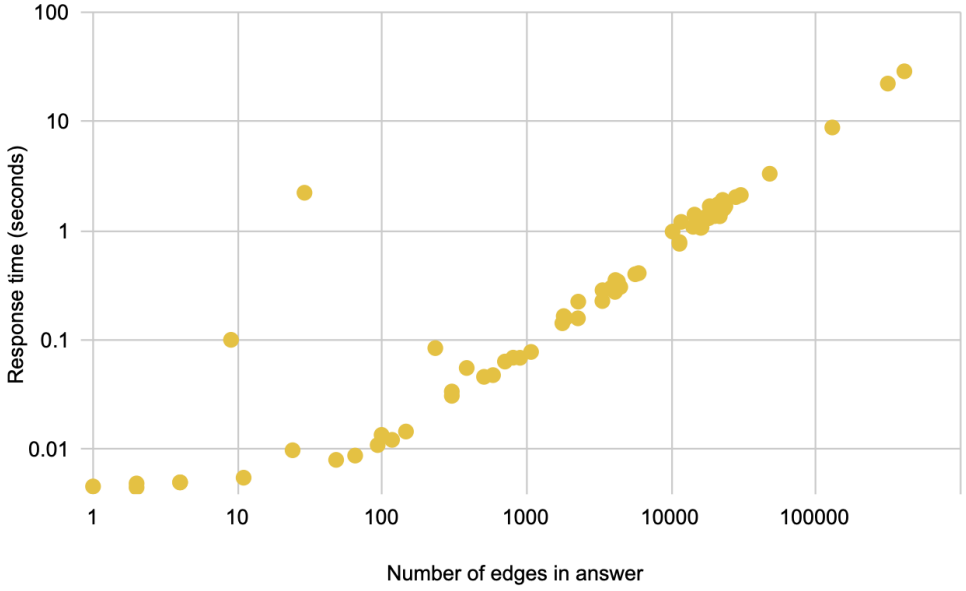

Figure 2: PloverDB's query response time vs. the number of edges in the response when hosting the RTX-KG2 knowledge graph.

it is relatively memory-hungry, using 79 GiB of system memory at rest when hosting the RTX-KG2.8.4c knowledge graph (7 million nodes and 27 million edges).

[4] `tiangolo/uwsgi-nginx-flask:python3.11`

[5] Available in the TRAPI GitHub repository, at ReasonerAPI/TranslatorReasonerAPI.yaml.

3 Usage

Full technical details on how to use and deploy PloverDB are provided in GitHub at github.com/RTXteam/PloverDB. Assuming that Docker has been installed on the host computer and the PloverDB code repository has been cloned into the local file system, deploying PloverDB is simply a matter of:

1. editing `PloverDB/app/config.json` for the user's KG, and
2. running `bash PloverDB/run.sh`, which builds and starts the Dockerized PloverDB service.

The PloverDB repository includes a template for the `config.json` file. Most significantly, users must update the `nodes_file` and `edges_file` slots with URLs from which PloverDB can download the graph's nodes and edges files at image build time. These files must adhere to Biolink KGX (github.com/biolink/kgx) TSV or JSON Lines format. Users may also specify how node/edge properties should be represented as TRAPI Attributes using the `trapi_attribute_map` slot. Explanations for all `config.json` properties are available in the PloverDB GitHub repository, as well as steps to deploy PloverDB in a production setting, including TLS certificate installation.

The fact that the `config.json` file is tracked in the PloverDB software repository makes configuring continuous deployment straightforward; users can set up continuous deployment on a fork of the repository using a third-party tool, or they can use PloverDB's built-in remote deployment mechanism, which allows users to deploy changes by submitting an authenticated `/rebuild` request to the PloverDB web service. Multiplexing KGs is a matter of adding an additional JSON config file for each KG to be served. PloverDB will automatically build and expose a TRAPI endpoint for each such KG, accessible at the `endpoint_name` specified in its configuration file.

At the time of writing, PloverDB is used to host and serve five different knowledge graphs for the Translator system: RTX-KG2 ([8]; kg2cploverdb.transltr.io/kg2c), Microbiome KG ([4]; multiomics.transltr.io/mbkp), Multiomics KG ([6]; multiomics.transltr.io/mokp), Clinical Trials KG ([6]; multiomics.transltr.io/ctkp), and Drug Approvals KG ([6]; multiomics.transltr.io/dakp), ranging in size from 5,000 nodes and 45,000 edges to 7 million nodes and 27 million edges.

4 Conclusion

PloverDB is a software platform for efficiently hosting and serving Biolink-compliant knowledge graphs as standardized TRAPI-compliant web APIs. PloverDB emphasizes simplicity and automation, allowing knowledge graph owners to focus on their data rather than the technicalities of exposing it. Its comprehensiveness in terms of built-in reasoning and ease of deployment significantly reduce the barrier of entry for exposing knowledge graphs for use by others, thereby enhancing their utility for advancing biomedicine.

5 Competing interests

No competing interest is declared.

6 Author contributions statement

A.K.G., S.A.R., and E.W.D. conceived the application. A.K.G. designed, implemented, and analyzed the software. A.K.G. and S.A.R. wrote the manuscript.

7 Acknowledgments

We thank David Koslicki, Gwênlyn Glusman, Mohsen Taheri, Kevin Vizhalil, Sundareswar Pullela, Skye Goetz, Chunyu Ma, Lindsey Kvarfordt, E. C. Wood, Evan Morris, and Max Wang for advice, inspiration, and/or feedback. We thank the NCATS Information Technology Resources Branch (ITRB) for incorporating PloverDB into their automated build and deployment system.

8 Funding

This work was supported by the National Institutes of Health, National Center for Advancing Translational Sciences [OT2TR003428, OT2TR002520]. Any opinions expressed in this document are those of the authors and do not

necessarily reflect the views of NIH, NCATS, other Translator team members, or affiliated organizations and institutions. A.K.G. gratefully acknowledges support from the ARCS Foundation.

References

[1] J. Callaghan, C. H. Xu, J. Xin, M. A. Cano, A. Riutta, E. Zhou, R. Juneja, Y. Yao, M. Narayan, K. Hanspers, A. Agrawal, A. R. Pico, C. Wu, and A. I. Su. BioThings Explorer: a query engine for a federated knowledge graph of biomedical APIs. *Bioinformatics*, 39(9):btad570, 2023. ISSN 1367-4811. URL `https://doi.org/10.1093/bioinformatics/btad570`.

[2] K. Fecho, A. Thessen, S. E. Baranzini, C. Bizon, J. J. Hadlock, S. Huang, R. T. Roper, N. Southall, C. N. Ta, P. B. Watkins, M. D. Williams, H. Xu, W. E. Byrd, V. Dancík, M. Duby, M. Dumontier, G. Glusman, N. L. Harris, E. W. Hinderer, G. Hyde, A. J. Johs, A. I. Su, G. Qin, and Q. Zhu. Progress toward a universal biomedical data translator. *Clinical and Translational Science*, 15:1838 – 1847, 2022. URL `https://doi.org/10.1111/cts.13301`.

[3] A. K. Glen, C. Ma, L. Mendoza, F. Womack, E. C. Wood, M. Sinha, L. Acevedo, L. G. Kvarfordt, R. C. Peene, S. Liu, A. S. Hoffman, J. C. Roach, E. W. Deutsch, S. A. Ramsey, and D. Koslicki. ARAX: a graph-based modular reasoning tool for translational biomedicine. *Bioinformatics*, 39, 2023. URL `https://doi.org/10.1093/bioinformatics/btad082`.

[4] S. L. Goetz, A. K. Glen, and G. Glusman. MicrobiomeKG: Bridging microbiome research and host health through knowledge graphs. *bioRxiv*, 2024. URL `https://doi.org/10.1101/2024.10.10.617697`.

[5] S. Lelong, X. Zhou, C. Afrasiabi, Z. Qian, M. A. Cano, G. Tsueng, J. Xin, J. Mullen, Y. Yao, R. Avila, G. Taylor, A. I. Su, and C. Wu. BioThings SDK: a toolkit for building high-performance data APIs in biomedical research. *Bioinformatics*, 38(7):2077–2079, 2022. ISSN 1367-4803. URL `https://doi.org/10.1093/bioinformatics/btac017`.

[6] G. Qin, K. Narsinh, Q. Wei, J. C. Roach, A. Joshi, S. L. Goetz, S. T. Moxon, M. H. Brush, C. H. Xu, Y. Yao, A. K. Glen, E. D. Morris, A. Ralevski, R. T. Roper, B. Belhu, Y. Zhang, I. Shmulevich, J. J. Hadlock, and G. Glusman. Generating biomedical knowledge graphs from knowledge bases, registries, and multiomic data. *bioRxiv*, 2024. URL `https://doi.org/10.1101/2024.11.14.623648`.

[7] D. R. Unni, S. T. Moxon, M. Bada, M. H. Brush, R. M. Bruskiewich, P. A. Clemons, V. Dancík, M. Dumontier, K. Fecho, G. Glusman, J. J. Hadlock, N. L. Harris, A. Joshi, T. Putman, G. Qin, S. A. Ramsey, K. A. Shefchek, H. R. Solbrig, K. Soman, A. Thessen, M. A. Haendel, C. Bizon, C. J. Mungall, and the Biomedical Data Translator Consortium. Biolink Model: A universal schema for knowledge graphs in clinical, biomedical, and translational science. *Clinical and Translational Science*, 15:1848 – 1855, 2022. URL `https://doi.org/10.1111/cts.13302`.

[8] E. Wood, A. K. Glen, L. G. Kvarfordt, F. Womack, L. Acevedo, T. S. Yoon, C. Ma, V. Flores, M. Sinha, Y. Chodpathumwan, A. Termehchy, J. C. Roach, L. Mendoza, A. S. Hoffman, E. W. Deutsch, D. Koslicki, and S. A. Ramsey. RTX-KG2: a system for building a semantically standardized knowledge graph for translational biomedicine. *BMC Bioinformatics*, 23, 2021. URL `https://doi.org/10.1186/s12859-022-04932-3`.

[9] A. Zaveri, S. Dastgheib, C. Wu, T. Whetzel, R. Verborgh, P. Avillach, G. Korodi, R. Terryn, K. Jagodnik, P. Assis, and M. Dumontier. smartAPI: Towards a more intelligent network of Web APIs. In E. Blomqvist, D. Maynard, A. Gangemi, R. Hoekstra, P. Hitzler, and O. Hartig, editors, *The Semantic Web*, pages 154–169, Cham, 2017. Springer International Publishing. ISBN 978-3-319-58451-5.

A Examples - queries and reasoning

To better illustrate the "one-hop" Translator Reasoner API (TRAPI) queries described in the main text that PloverDB is designed to answer, we provide some example query graphs below. They are represented in an abbreviated form, rather than full TRAPI JSON[6].

1. `(BRCA1)-[associated_with]->(breast cancer)`
2. `(BRCA1)-[associated_with]->(Disease)`

[6] Black text represents specific "pinned" concepts, green text represents edge type constraints, gold text represents node type constraints, purple text represents edge attribute constraints, and asterisks represent a "wildcard" node/edge type. In reality, nodes are referred to only by standard identifiers, rather than their names (e.g., `NCBIGene:672` for the gene *BRCA1*).

3. `(Gene)-[associated_with]->(cancer)`
4. `(BRCA1, BRCA2)-[associated_with]->(ChemicalEntity)`
5. `(Drug, SmallMolecule)-[treats]->(breast cancer)`
6. `(BRCA1)-[associated_with, affects]->( * )`
7. `(BRCA1)-[ * ]->( * )`
8. `(BRCA1)-[ * , source_type != text_mined]->( * )`

As explained in the main text, when answering such query graphs, PloverDB performs some knowledge reasoning tasks beyond direct pattern-matching. To better explain these, we have included a few examples below (based off of the above example queries).

- *Predicate symmetry/asymmetry*: When answering example Query 1, PloverDB ignores edge direction (i.e., it effectively also looks for patterns in the opposite direction), since the edge type `associated_with` is annotated as symmetric in the Biolink model. When answering Query 5, however, it will obey edge direction, since `treats` is an asymmetric predicate according to Biolink.
- *Concept subclasses*: When answering Query 3, PloverDB returns genes associated with the general concept of "cancer" as well as genes associated with any subclasses of cancer (e.g., breast cancer, pancreatic cancer, etc.).
- *Node category hierarchy*: When answering Query 4, PloverDB will return any chemical entities associated with *BRCA1* or *BRCA2*, as well as any small molecules, drugs, and other nodes with categories that are descendants of `ChemicalEntity` in the Biolink Model, that are associated with either of those genes.
- *Edge predicate hierarchy*: When answering Query 1, PloverDB will look not only for edges linking *BRCA1* and breast cancer that have the type `associated_with`, but also any descendant of that edge type in the Biolink Model, such as `correlated_with`, `gene_associated_with_condition`, etc.

While a full description of how TRAPI queries should be constructed is available in the TRAPI GitHub repository (github:NCATSTranslator/ReasonerAPI), we provide an example of a TRAPI query expressed in full JSON format below. This query submits our example Query 2 to the PloverDB-hosted RTX-KG2 API.

```
curl -X 'POST' \
  'https://kg2cploverdb.transltr.io/query' \
  -H 'accept: application/json' \
  -H 'Content-Type: application/json' \
  -d '{
  "message": {
    "query_graph": {
      "nodes": {
        "n00": {
          "ids": [
            "NCBIGene:672"
          ]
        },
        "n01": {
          "categories": [
            "biolink:Disease"
          ]
        }
      },
      "edges": {
        "e00": {
          "subject": "n01",
          "object": "n00",
          "predicates": [
            "biolink:associated_with"
          ]
        }
      }
    }
```

```
  }
}'
```

B Methods for performance analysis

Methodology details for the controlled assessment of PloverDB's performance when hosting the RTX-KG2 knowledge graph are provided in the following sections.

B.1 Platforms and hosting

PloverDB was run in Docker (Debian 11.11 base image) to host the RTX-KG2 knowledge graph on a dedicated Amazon Web Services (AWS) Elastic Compute Cloud (EC2) `r5a.4xlarge` instance (128 GiB system memory, Ubuntu Linux version 18.04[7]). Version 2.8.4 of RTX-KG2 was used; we used the canonicalized version of the graph (RTX-KG2.8.4c), which had 7 million nodes and 27 million edges. We used the version of RTX-KG2.8.4c in which certain edges with questionable confidence[8] had been filtered out. The version of PloverDB used in this empirical study supported TRAPI v1.4.

To best simulate real-world usage, test queries were sent via HTTP requests (without caching) from a separate EC2 instance (also an `r5a.4xlarge`) in the same AWS Region as the hosting platform (`us-west-2`).

B.2 Selection of test queries

To generate a sample of real-world queries to evaluate PloverDB's performance on, we randomly selected a set of 90 queries out of those submitted to live (publicly available web service endpoints registered via SmartAPI) RTX-KG2 knowledge provider services (which include development, staging, test, and production instances) between Dec. 6, 2023 and March 6, 2024. To cover a variety of RTX-KG2 use cases yet ensure queries to production instances (which are relatively less common) are still represented, we divided this sampling into three subsets, as follows:

- "**Any**": 30 queries randomly selected from the overall set of 11,234 unique[9] RTX-KG2 queries submitted to any RTX-KG2 instance (development, staging, etc.) during the 3-month window
- "**ITRB prod**": 30 queries randomly selected from the subset of 59 unique queries sent to the production RTX-KG2 instance hosted by the NCATS Information Technology Resources Branch (ITRB) during the 3-month window
- "**Long**": 30 queries randomly selected from the subset of 351 queries with an elapsed time over 30 seconds during the 3-month window

For the first two of these samplings, we excluded queries with an elapsed time[10] under two seconds to bias the sample towards queries that actually produce answers (answer size is not recorded for past RTX-KG2 queries), since we were most interested in analyzing speed for queries that RTX-KG2 can actually answer. In addition, we included three hand-crafted queries designed to produce very large answer sets in comparison to typical queries of RTX-KG2 (i.e., >300k edges) to address gaps in the analysis in terms of answer size. The final 93 queries are available on GitHub at github:amykglen/plater-plover/tree/main/test. Of these 93 queries, 67 produced a non-zero answer from the PloverDB-hosted RTX-KG2 and were included in the final analysis.

[7] Host operating system and processor architecture combinations that have been verified to successfully run the Dockerized PloverDB include Ubuntu/x86_64 (18, 20, and 22) and macOS/arm64 (20, 21, and 23). The fact that PloverDB is fully Dockerized means that it should be straightforward to run on any host operating system.

[8] Defined as edges that either 1) came from SemMedDB and have fewer than four supporting publications, or 2) had incompatible subject and object node categories according to Biolink.

[9] Queries were deduplicated by creating hash keys for each query that consisted of a string somewhat similar to a Neo4j Cypher query, in which node IDs were canonicalized and sorted, categories/predicates were sorted, and qualifiers were included as well. The deduplication method is available in the `sample_kg2_queries.py` script in the RTX GitHub repository.

[10] These times were those taken by the previous version of the RTX-KG2 API software stack (prior to PloverDB's conversion to a standalone TRAPI web service), which was not evaluated in this study.

B.3 Data collection/analysis methods

We ran three repetitions of the full test suite of 93 queries. Queries were run in random order on each run of the test suite[11]. Between those three runs, the host instance was rebooted and PloverDB was rebuilt to clear any cached data (in the operating system or elsewhere). The testing instance was also rebooted between the three runs. Query duration was measured as elapsed wall clock time from the beginning of the query until the response *began* to be streamed back to the testing instance (i.e., when the response headers were returned).

System memory usage was obtained using the `psutil`[12] Python library's `virtual_memory` method; the `percent` value returned from this method was collected immediately after the PloverDB build completed, prior to the submission of any queries (i.e., when the platform was "at rest"). Percentages were then converted to GiB values by calculating $(percent/100) * total$, where `total` is the instance's total physical memory according to `psutil.virtual_memory` (124.7 GiB for the host instance used). Memory usage was averaged across three build repetitions.

Analyses were done in Python using Pandas (v2.0.3) and SciPy (v1.11.4) and in Google Sheets. Testing was done in April 2024. Results are archived at github:amykglen/plater-plover/blob/main/test/final_results/results_plovernote.tsv.

[11]Technically the four subsets of queries were always run in the same order (ITRB Prod, Any, Long, Hand-crafted), but within those subsets queries were run in random order.

[12]psutil.readthedocs.io