Design and Benchmarking of a Citation Graph DB Across Neo4j, ArangoDB, and MASS Graph DB Systems

Yumeng Pang

A thesis submitted in partial fulfillment of the
requirements for the degree of

Master of Science

University of Washington Bothell

2026

Reading Committee:

Munehiro Fukuda, Committee Chair
Johnny Lin
Annuska Zolyomi

Program Authorized to Offer Degree:

Computer Science & Software Engineering

University of Washington

Abstract

Design and Benchmarking of a Citation Graph DB Across Neo4j, ArangoDB, and MASS Graph DB Systems

Yumeng Pang

Chair of the Supervisory Committee:
Munehiro Fukuda
Computer Science & Software Engineering

Academic collaboration, citation influence, and institutional research visibility are increasingly reflected through scholarly relationship networks. However, existing academic platforms remain largely profile-centered and do not provide an institution-focused, interactive, and queryable graph system for multi-hop exploration across authors, works, affiliations, and citations. This research investigates the design and benchmarking of a UWB citation graph, seeded from CSS faculty scholarly activities for practical evaluation, and examines how effectively different graph database systems support this richer graph model for practical scholarly exploration.

To address this problem, this work designs and implements a scholarly citation and co-authorship graph pipeline that constructs a heterogeneous Author–Work–Citation–Affiliation graph using institutional seed data and OpenAlex-derived metadata. The resulting graph is intended to support practical use cases such as collaborator discovery and referee exploration for UWB CSS faculty. The system is evaluated across three graph databases—Neo4j, ArangoDB, and MASS Graph DB—and is benchmarked using LDBC-aligned workloads and metrics, including bulk ingestion throughput, query throughput, and multi-hop traversal latency. In addition to the institutional citation graph, the evaluation framework includes public benchmark datasets of different graph types, densities, and scales to enable broader cross-platform comparison.

The results support the hypothesis that a heterogeneous scholarly citation graph can enable richer institution-centered analysis while maintaining practical performance. Compared with a single-relation citation graph, the UWB-CSS Citation Graph supports collaborator discovery, citation-based visibility analysis, affiliation-aware filtering, and preliminary external-referee identification. Although its heterogeneous author–work–affiliation–citation structure introduces additional ingestion and traversal cost, its sparse topology helps offset part of the performance cost introduced by heterogeneous node and relationship types. Platform-specific results further show that Neo4j provides strong interactive query and traversal performance for the UWB-CSS graph, MASS Graph DB supports high-throughput in-memory bulk ingestion, and ArangoDB Cloud offers useful managed deployment and distributed scaling potential, especially for bulk ingestion and 3-hop traversal from 4 to 8 nodes. Overall, the proposed graph remains practical on suitable platforms while providing broader analytical value than structurally simpler citation-only graphs.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

ACKNOWLEDGMENTS

I would like to express my sincere gratitude to Professor Munehiro Fukuda for his guidance, support, and encouragement throughout this research. I also thank Professor Johnny Lin and Professor Annuska Zolyomi for serving on my committee and for their valuable feedback. In addition, I am grateful to the University of Washington Bothell and the Computer Science & Software Engineering program for providing the academic environment and support that made this work possible.

Chapter 1

INTRODUCTION

This research aims to design, construct, and benchmark an institution-centered scholarly citation graph that helps UWB CSS faculty explore research relationships across authors, works, affiliations, and citations while evaluating which graph database systems best support this heterogeneous workload. This chapter introduces the motivation, research objectives, and central hypothesis of this thesis by explaining why a UWB CSS faculty–seeded citation graph is needed, what this work aims to build and evaluate, and how the proposed graph may benefit faculty and broader scholarly analysis.

1.1 Motivation

Existing scholarly platforms provide useful access to researcher profiles, publication records, and citation information, but they often remain limited to profile-centered or paper-centered exploration. In academia, researchers frequently need to understand not only who published a paper, but also how authors, works, institutions, collaborations, and citations are connected across multiple relationship paths. However, many existing systems do not provide an institution-centered, queryable, and graph-native structure that supports flexible multi-hop exploration across these scholarly relationships.

A graph-based scholarly system is therefore useful because academic collaboration and research influence are naturally relational. Researchers may want to identify potential collaborators who work in related areas, discover scholars who repeatedly cite their publications, or locate external experts who are familiar with their research but do not have close institutional or recent co-authorship conflicts. By modeling authors, works, affiliations, and citations together, a heterogeneous citation graph can support these practical academic needs more effectively than isolated publication lists or citation-only networks.

At the same time, building a richer scholarly graph introduces important system-level

questions. A heterogeneous scholarly graph provides more analytical value than a simpler single-relation citation graph, but it may also increase graph ingestion cost, meaning the cost of inserting or materializing graph nodes and relationships into a database backend, along with query complexity, storage overhead, and multi-hop traversal latency. This motivates the need to benchmark the proposed academic citation graph across multiple graph database systems in order to evaluate whether its added scholarly expressiveness can be supported with practical performance.

1.2 Research Hypothesis

A central hypothesis of this thesis is that a heterogeneous scholarly citation graph integrating authorship, citation, and affiliation relationships can support richer institution-centered scholarly analysis than structurally simpler single-relation citation graphs while still maintaining practical interactive query and traversal performance.

This research extends the traditional citation graph from a paper-to-paper citation network into a heterogeneous scholarly citation graph. In addition to preserving citation relationships between works, the proposed graph also models authors, affiliations, and authorship links. In this research, "richer" scholarly analysis refers to the graph's ability to preserve and query multiple kinds of scholarly context rather than only one relationship type. Specifically, richness is evaluated qualitatively through five dimensions: node-type diversity, relationship-type diversity, query expressiveness, contextual filtering capability, and supported scholarly use cases. A single-relation citation graph can represent paper-to-paper citation links, but it cannot directly represent who authored each paper, which institution each author belongs to, whether two scholars recently collaborated, or whether a citing scholar is institutionally independent from the target faculty member. In contrast, the proposed heterogeneous graph supports cross-relational exploration across authors, works, affiliations, authorship, affiliation, and citation paths, making it more suitable for institution-centered tasks such as collaborator discovery, citation-based visibility analysis, and preliminary external referee identification.

1.3 Research Objectives

The first objective of this research is to design and implement a heterogeneous scholarly citation graph that can be used as a general model for institution-centered scholarly relationship analysis. As a concrete case study, this research uses UWB CSS faculty scholarly activities as the seed data and constructs the graph from OpenAlex metadata. The graph represents scholarly information through a shared Author–Work–Affiliation schema, where researchers, publications, institutions, authorship relationships, affiliation links, and citation connections are modeled together in one structure. By using UWB CSS as an example implementation, the research demonstrates how this general graph model can support richer scholarly exploration than a citation-only network.

The second objective is to apply the heterogeneous scholarly citation graph to three graph database systems: Neo4j, ArangoDB, and MASS Graph DB. This cross-platform deployment is used to evaluate which system features best support different parts of citation graph processing, including graph loading, bulk ingestion, interactive querying, multi-hop traversal, and distributed scalability. Through this comparison, the research aims to identify the platform-specific strengths and trade-offs of using each system for scholarly graph workloads.

The third objective is to compare the heterogeneous scholarly citation graph with other citation graph and benchmark graphs of different types, sizes, and densities. This comparison is used to evaluate whether the proposed scholarly citation graph is qualitatively and quantitatively competitive with existing graph datasets while providing broader analytical value through its heterogeneous structure. By comparing performance across graph topologies, the research examines whether the proposed citation graph remains practical while enabling richer academic use cases such as collaborator discovery and referee-candidate exploration.

1.4 Research Impact

The primary stakeholders are faculty researchers, who can benefit from a graph-based system that helps identify potential collaborators, understand citation-based research visibility, and

locate external scholars who may be suitable promotion or tenure referee candidates.

More broadly, this research can benefit researchers and system designers interested in scholarly graph analytics by showing how heterogeneous citation graphs can be constructed, validated, and benchmarked across different graph database systems.

Chapter 2
BACKGROUND

This chapter provides the background needed to understand the graph database platforms evaluated in this research, including MASS, MASS Graph DB, previous MASS Graph DB implementations, Neo4j, and ArangoDB, with emphasis on the system features that affect later benchmarking interpretation.

2.1 MASS

MASS (Multi-Agent Spatial Simulation library) was developed to reduce the semantic gap between entity-centered simulation models and lower-level parallel programming frameworks such as OpenMP, MPI, and MapReduce by expressing distributed computation through two core abstractions: Places and Agents [1]. In this design, Places form a multi-dimensional array of network-independent elements distributed across a cluster of multi-core machines, while Agents represent active execution instances that can reside on a place, migrate to other places, interact with both places and peer agents, and encapsulate computation at the level of individual entities rather than explicit loop-based coordination. MASS executes these abstractions over a set of communicating processes connected through SSH-tunneled TCP links, with the runtime handling thread creation, communication, and scheduling; places are typically partitioned statically across processes or threads, whereas agents are scheduled more dynamically according to their proximity to the places maintained by each process. Collective operations such as callAll, exchangeAll, and manageAll further allow the runtime to coordinate parallel method invocation, data exchange, and agent-state management across distributed memory, making localized entity interaction the fundamental execution model of the system [1].

MASS is relevant to graph database research because prior work demonstrated that dispatching agents over a graph stored in distributed memory can outperform conventional

graph-streaming-style processing for repeated graph analysis. The pipelining study shows that MASS overlaps graph construction with agent-based computation, achieving up to 7.7× speedup over non-pipelined single-node execution and 8.3× speedup from one to twenty-four pipelined nodes, which motivates applying MASS as the execution foundation for an in-memory, agent-based graph database system [2].

2.2 MASS Graph Database

MASS Graph DB extends the MASS framework into a distributed, in-memory graph database model by storing graph data in memory and dispatching agents as query executors over the graph. In this design, Cypher-style query logic is translated into agent behavior: agents are deployed over the graph, traverse vertices and relationships, clone when a query path branches, collect path results, and return matching graph patterns through MASS execution cycles. Therefore, MASS Graph DB differs from disk-persistent systems such as Neo4j and ArangoDB because its primary execution model is agent-based graph traversal over distributed memory rather than centralized query execution over a persistent storage engine [3].

Previous MASS Graph DB research progressively transformed MASS from a general agent-based runtime into a prototype distributed graph database. Rajvaidya [4] first extended GraphPlaces with generic CSV-driven graph construction, VertexPlace-based storage of graph nodes and edges, an ANTLR/OpenCypher parsing pipeline that translated MATCH-style queries into MASS method calls, and baseline WHERE, aggregate, and multi-relationship traversal queries for correctness and latency/scalability testing. Later work moved the system toward a property-graph design by storing graph data in PropertyVertexPlace and PropertyGraphPlaces, dispatching graph agents over distributed in-memory vertices, and benchmarking common depth-1 to depth-3 traversal workloads against Neo4j and ArangoDB on randomized graphs up to 30K nodes, which clarified both the graph-loading advantages and traversal trade-offs of the agent-based approach [5], [6]. Prajapati [6] further expanded query expressiveness by adding Cypher WHERE-clause filtering through abstract-syntax-tree construction, execution-plan building, and agent-side constraint evaluation.

In this research, MASS Graph DB is evaluated as a research-oriented graph processing platform rather than a fully persistent production database system. This distinction is important because MASS provides useful support for in-memory graph construction, agent-based traversal, and benchmark comparison, but it does not yet provide the same level of disk persistence, transaction management, or production-ready multi-node interactive query and traversal support.

2.3 Neo4j

Neo4j is used in this research as a property-graph baseline because its data model represents information as labeled nodes, directed relationships, and key-value properties, which fits the general structure of a heterogeneous scholarly citation graph. Its Cypher query language [7] is also relevant because declarative pattern matching can express scholarly relationship queries, including publication paths, citation links, collaboration neighborhoods, and affiliation-based filtering. Neo4j further supports indexes and uniqueness constraints, which are useful for enforcing entity consistency and improving lookup efficiency during graph loading and query execution [7]. At the system level, Neo4j should be interpreted mainly as a mature single-server and read-oriented baseline because its scalability is strongest through vertical scaling and clustered read/availability support rather than full distributed graph partitioning [8]. Therefore, in this thesis, Neo4j serves as the main baseline for latency-sensitive scholarly exploration, where its index-free adjacency model [9] and mature query engine are expected to support strong query throughput and efficient multi-hop traversal performance.

2.4 ArangoDB

ArangoDB is used in this research as a multi-model, cloud-oriented graph database baseline because it supports graph, document, and key-value data models within one system, while its graph model stores vertices in document collections and relationships in edge collections with _from and _to references [10]. This document-oriented property graph design makes ArangoDB suitable for representing a heterogeneous scholarly citation graph in which scholarly entities and relationships can be stored as flexible JSON-like documents

and queried through ArangoDB Query Language (AQL) [10]. In a cluster deployment, ArangoDB operates through coordinators, DB-Servers, and sharded collections, allowing data to be horizontally distributed and rebalanced across multiple servers [11]. This makes ArangoDB useful for evaluating whether the citation graph can benefit from distributed cloud deployment; however, the results must be interpreted through the trade-off between horizontal scalability and coordination overhead, because graph traversals over randomly sharded or poorly co-located data may require network hops across servers, while optimized graph sharding strategies such as SmartGraphs or EnterpriseGraphs are designed specifically to reduce such cross-server traversal costs [12].

2.5 Summary of Key Benchmarking and System Terms

Because this thesis compares graph databases with different storage models, execution mechanisms, and workload behavior, several key terms are used consistently throughout the evaluation. Graph density refers to the ratio of actual directed edges to the maximum possible number of directed edges in a graph, and it helps explain how many relationships may be encountered during ingestion, querying, and traversal. A 3-hop traversal refers to a bounded graph exploration that follows three consecutive relationships from a starting node, which approximates neighborhood-style scholarly exploration such as moving from an author to works, citations, or related authors. Bulk ingestion measures the throughput of loading many graph elements in large batches, while incremental ingestion measures smaller update operations that approximate gradual graph growth. Query throughput, reported as queries per second, measures how many lightweight read queries a system can complete within a fixed time interval. Latency measures the response time of an individual query or traversal. Platform-specific persistence refers to whether a system stores graph data durably as part of a database engine, as in Neo4j and ArangoDB, or materializes it primarily in memory, as in the current MASS Graph DB implementation. In-memory execution refers to graph processing performed mainly over memory-resident graph structures, which can improve loading speed but may not provide the same persistence as a production database. Cypher refers to Neo4j's declarative graph query language, used in this thesis to express graph pattern-matching queries, while AQL refers to ArangoDB's query language for document

and graph traversal operations. These terms provide the common vocabulary for interpreting the benchmark results, where differences in density, traversal depth, ingestion method, persistence model, query language, and deployment configuration all affect the observed performance trade-offs among Neo4j, ArangoDB, and MASS Graph DB.

Chapter 3

RELATED WORK

This chapter reviews prior work that motivates the design and evaluation of a heterogeneous scholarly citation graph. The discussion first examines scholarly networking platforms and graph-based literature exploration systems, then reviews prior studies on scholarly relationship modeling, citation and co-authorship analysis, graph database benchmarking, and the specific research gap addressed by this research.

3.1 Scholarly Networking Platforms and the Need for Queryable Relationship Exploration

Academic platforms such as ResearchGate and Academia.edu demonstrate the need for systems that help researchers maintain scholarly profiles, share publications, and increase research visibility [13], [14]. However, these platforms are primarily profile and publication centered, so they do not fully support graph-native, multi-hop exploration across authors, works, affiliations, co-authorship links, and citation paths.

This limitation matters because many scholarly discovery tasks are naturally relational, such as identifying scholars who frequently cite a researcher's work, finding potential collaborators in related areas, or filtering referee candidates by affiliation and recent collaboration history. Therefore, instead of building another profile-centered scholarly network, this research constructs a heterogeneous scholarly citation graph for institution-centered exploration that supports multi-hop exploration.

3.2 Scholarly Relationships as Graph Structures: Citation, Co-authorship, and Research Communities

Scholarly relationships are naturally graph-structured because authors, publications, collaborations, and citations form connected networks that can reveal research proximity, influence, and community structure. Prior work shows that citation and author collab-

oration networks can be explored using graph databases [15], academic social-network data can support collaborator recommendation [16], co-authorship and citation relationships are meaningful signals for understanding scholarly influence [17], and authorship networks can be used to examine research communities across scientific domains [18].

However, much of this prior work studies citation, co-authorship, or authorship networks separately rather than integrating authors, works, affiliations, and citation links into one heterogeneous graph. This research builds on those findings by designing a heterogeneous scholarly citation graph that combines citation, co-authorship, and affiliation signals to support richer institution-centered tasks such as collaborator discovery and preliminary referee-candidate exploration.

3.3 Graph-Based Scholarly Exploration and Large-Scale Querying

Graph-based scholarly exploration systems show that research discovery can benefit from representing literature as connected structures rather than isolated search results. Connected Papers supports visual exploration of related papers around a seed work [19], and Behera et al. further show that this type of graph-based navigation can help researchers understand nearby literature clusters and topic relationships [20].

Large-scale scholarly graph systems also demonstrate that academic metadata can be queried through graph database infrastructure. GrapAL, for example, uses a Neo4j-backed graph to connect scientific literature at scale, showing the feasibility of graph-native scholarly querying [21]. However, existing tools are often paper-level, visualization-focused, or tied to a single backend, so they do not fully address institution-centered scholarly exploration and cross-platform backend suitability. This motivates both the exploration use case of this research and the need to evaluate how Neo4j, ArangoDB, and MASS Graph DB support the same heterogeneous scholarly citation graph.

3.4 Open Scholarly Metadata for Reproducible Graph Construction

Open scholarly metadata sources make it possible to construct citation and collaboration graphs in a reproducible way because they provide machine-readable records for authors, works, institutions, identifiers, authorship, and citation relationships. OpenAlex is espe-

cially useful for this research because it exposes structured scholarly metadata through an open API and supports persistent identifiers such as ORCID, OpenAlex IDs, and DOI values, which help normalize entities and reduce duplicate authors or works during graph construction [22].

However, open metadata sources may still contain missing publications, incomplete affiliations, or ambiguous author identities, so automated collection must be paired with identifier normalization and manual validation. In this research, OpenAlex provides the main metadata foundation for building the heterogeneous scholarly citation graph, while manual checking improves correctness for faculty identity, publication completeness, and misplaced author-work associations.

3.5 Graph Database Benchmarking Foundations and the Need for Scholarly Workload Evaluation

Graph database benchmarking provides a structured way to compare systems using measurable workload categories such as ingestion throughput, query throughput and multi-hop traversal performance. The LDBC (Linked Data Benchmark Council) Social Network Benchmark is especially relevant because it provides a widely recognized graph benchmarking reference for evaluating database behavior under social-network-style workloads, which makes it useful as a baseline for comparing graph platforms under controlled conditions [23].

However, general-purpose graph benchmarks do not fully capture the structure of scholarly workloads, where queries often combine citation paths, authorship links, affiliation filters, and researcher-centered exploration. Prior MASS Graph DB benchmarking work compared MASS with Neo4j and ArangoDB and showed the importance of backend-specific trade-offs [5], but this research extends that direction by evaluating a heterogeneous scholarly citation graph across multiple platforms and comparing its behavior with graphs of different types, scales, and densities.

3.6 Research Gap Addressed by This Thesis

Existing work shows that scholarly platforms, citation exploration tools, open scholarly metadata, and graph databases each address part of the scholarly discovery problem. How-

ever, these systems are often profile-centered, paper-centered, visualization-focused, or tied to a single backend, and they do not fully integrate authors, works, affiliations, authorship relationships, and citation links into one institution-centered heterogeneous scholarly graph.

This research addresses that gap by designing and constructing a heterogeneous scholarly citation graph that combines citation, co-authorship, and affiliation signals for richer scholarly exploration. Using UWB CSS faculty as a concrete case study, the graph supports practical academic tasks such as collaborator discovery, citation-based research visibility analysis, and preliminary identification of external promotion-referee candidates.

This research also addresses a benchmarking gap by evaluating the same scholarly graph across Neo4j, ArangoDB, and MASS Graph DB, while comparing it with graphs of different types, scales, and densities. This makes it possible to study not only whether the proposed graph is analytically useful, but also which backend systems best support ingestion, query throughput, and multi-hop traversal for the heterogeneous scholarly citation workloads.

Chapter 4

METHODOLOGY

This chapter describes the methodology used to construct and evaluate the UWB-CSS citation graph, including the data collection pipeline, normalization process, schema design, backend-specific graph loading, benchmarking framework, and validation methods.

4.1 System Architecture Overview

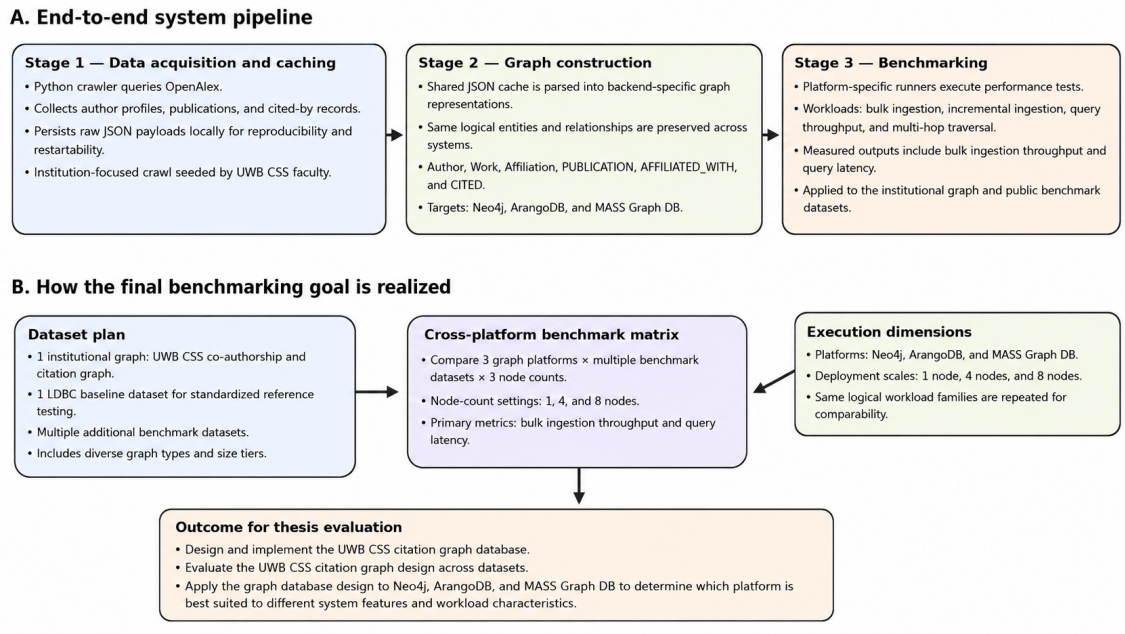


Figure 4.1: End-to-end pipeline overview of the crawler, backend-specific graph construction, and benchmarking workflow.

The overall system follows the pipeline architecture shown in Figure 4.1. Part A presents the end-to-end workflow of the system. The first stage is data acquisition and caching, where a Python crawler queries OpenAlex for author profiles, publications, and cited-by

data and stores the returned JSON payloads locally. The second stage is graph construction, where the shared JSON cache is transformed into Neo4j, ArangoDB, and MASS-specific graph representations while preserving the same logical entities and relationships. The third stage is benchmarking, where platform-specific runners execute bulk-ingestion, incremental-ingestion, query-throughput, and multi-hop traversal tests on both the institutional graph and public benchmark datasets. Part B extends this pipeline into the final evaluation design. It shows how the outputs of Part A, namely the constructed graphs and benchmarking workloads, are organized into a cross-platform benchmark matrix spanning multiple datasets, three platforms, and multiple node-count settings. In this sense, Part A describes how the system is built and executed, while Part B shows how that same pipeline is scaled into the broader comparative study used to evaluate performance trade-offs across graph types, dataset sizes, densities, and deployment configurations.

This architecture directly supports the thesis hypothesis by connecting the construction of a heterogeneous scholarly citation graph with a controlled cross-platform and cross-dataset evaluation process. By collecting authorship, citation, and affiliation data into one shared graph model, the methodology makes it possible to test whether the graph provides richer scholarly analysis than simpler citation-only or single-relation graph structures. At the same time, by loading the same logical graph into Neo4j, ArangoDB, and MASS Graph DB, and benchmarking it against datasets with different topology, scale, density, and relationship structure, the methodology evaluates whether the proposed graph remains practical in terms of ingestion throughput, query throughput, query latency, and multi-hop traversal performance. Therefore, the overall architecture is not only a data-processing pipeline, but also the experimental framework used to determine whether the proposed graph can achieve greater analytical expressiveness without unacceptable performance degradation across different platforms and graph structures.

4.2 Data Collection and Normalization

This section describes how the institutional citation graph is constructed from OpenAlex metadata as can be seen in Figure 4.2. The process begins with a manually curated seed set of UWB CSS faculty identifiers, including verified ORCID and OpenAlex author IDs, and

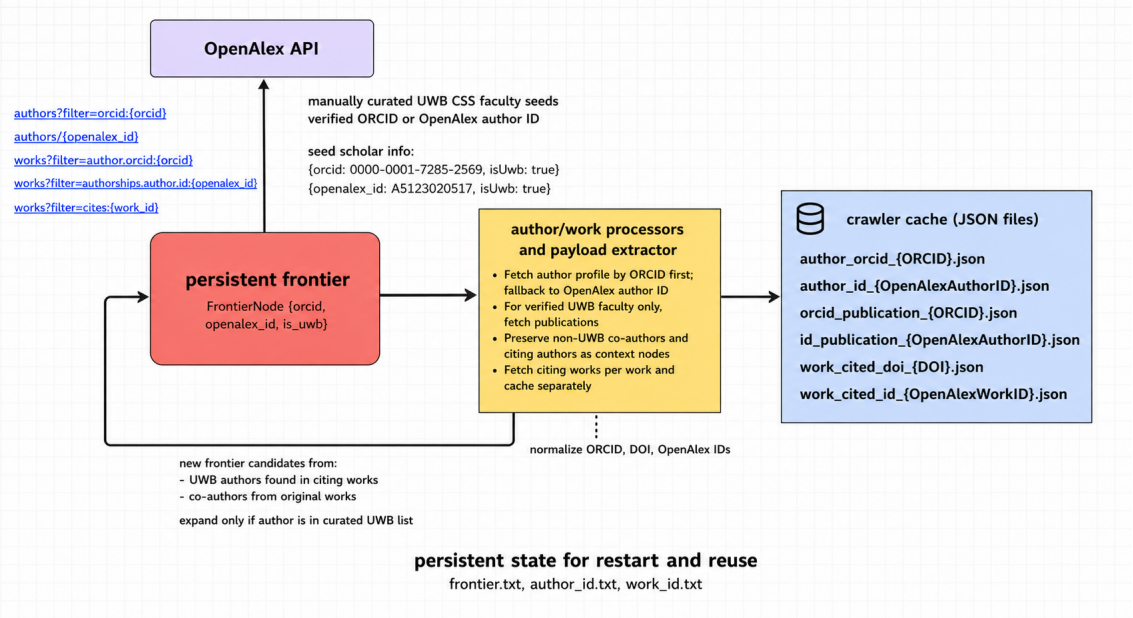


Figure 4.2: UWB CSS Faculty Related Scholar Info Crawler Flow Chart

uses a persistent crawler to retrieve author, publication, and citation data. The collected records are normalized and stored in a shared JSON cache, which serves as a backend-independent intermediate representation for subsequent loading into Neo4j, ArangoDB, and MASS Graph DB. Together, these steps establish a reproducible pipeline for building an institution-centered heterogeneous citation graph that supports authorship, affiliation, and citation analysis.

The following are the data collection and normalization process implemented sequentially in the crawler:

1. Data collection begins from a manually curated seed list of UWB CSS faculty. Each seed scholar is identified by a verified ORCID identifier when available, and otherwise by a verified OpenAlex author ID. These identifiers are inserted into the crawler's initial frontier so that the crawl starts from known faculty records and can resolve each seed author through the OpenAlex author API.

2. The crawler stores retrieved OpenAlex payloads as reusable JSON files organized by

persistent identifiers. For author profiles, it uses `author_orcid_{ORCID}.json` when the scholar is identified by ORCID, and `author_id_{OpenAlexAuthorID}.json` when the scholar is identified only by an OpenAlex author ID. For publication lists, it uses `orcid_publication_{ORCID}.json` when the source author is keyed by ORCID, and `id_publication_{OpenAlexAuthorID}.json` when the source author is keyed by OpenAlex author ID. For cited-work files, it uses `work_cited_doi_{DOI}.json` when the focal work has a DOI, and otherwise falls back to `work_cited_id_{OpenAlexWorkID}.json` when only an OpenAlex work ID is available. Across these files, the crawler preserves author metadata, publication metadata, authorship lists, institutional affiliation information, and citing-work records for later graph loading.

3. The crawler uses a persistent frontier to manage which scholars still need to be processed. When a frontier node is visited, the crawler fetches the author profile, verifies whether the scholar belongs to the curated UWB CSS faculty set, and expands publications only for verified UWB faculty. Co-authors from publication authorship lists are retained as context nodes, but recursive expansion proceeds only for authors whose ORCID or OpenAlex ID matches the UWB seed set. Likewise, citing works are fetched for each processed publication, and the resulting payloads are stored separately so that useful OpenAlex author, work, authorship, and affiliation fields can be reused by downstream loaders.

4. Identifier normalization and de-duplication are enforced before records are reused. OpenAlex author and work identifiers are converted from full URL forms into a consistent internal identifier format by removing the `https://openalex.org/` prefix and retaining only the stable identifier value. ORCID identifiers are normalized into a standard hyphenated form, and DOI values are reduced to a consistent canonical DOI string. The crawler then records these normalized identifiers in its persistent author and work ID sets so that repeated records, URL variants, and overlapping payloads are recognized as referring to the same Author, Work, and Affiliation entities across the cache and the downstream database loaders.

5. During graph construction, cached author records are transformed into `Author` nodes, publication and cited-work records are transformed into `Work` nodes, and institution records are transformed into `Affiliation` nodes. Authorship lists generate `PUBLICATION` edges from authors to works, affiliation records generate `AFFILIATED_WITH` edges from authors to affiliations, and citation payloads generate directed `CITED` edges from each citing work to the cited work.

6. Reproducibility is supported by the crawler's persistent state files and reusable cache. The frontier is stored on disk and rewritten after each pop or extension, while processed author identifiers and work identifiers are also persisted. As a result, the crawl can be stopped and resumed without losing state, repeated without reprocessing previously normalized records, and reused as a common input source for Neo4j, ArangoDB, and MASS Graph DB loaders.

4.3 Schema Design

Across platforms, the same logical graph model is used, as summarized in Table 4.1. The heterogeneous scholarly citation graph contains three node types: Author, Work, and Affiliation. The Author node represents a researcher and stores identity-related attributes such as name, ORCID, OpenAlex author ID, and faculty-related indicators when applicable. The Work node represents a scholarly publication and stores publication metadata such as DOI, title, publication year, type, and cited-by count. The Affiliation node represents an institution and stores institutional information used for affiliation-aware filtering. These nodes are connected through three relationship types: PUBLICATION, AFFILIATED_WITH, and CITED. PUBLICATION connects an Author to a Work to represent authorship, AFFILIATED_WITH connects an Author to an Affiliation to represent institutional membership, and CITED connects one Work to another Work to preserve citation direction from the citing publication to the cited publication. This schema is intentionally richer than a citation-only network because it allows scholarly relationships to be traversed across authors, works, institutions, and citations, supporting use cases such as collaborator discovery, citation-based influence analysis, and institution-aware referee-candidate filtering.

Table 4.1: Shared logical schema for the institutional citation graph.

Component type	**Name**	**Purpose in the graph model**
Node	`Author`	Represents a researcher and stores identifiers such as OpenAlex ID and ORCID.
Node	`Work`	Represents a publication and stores bibliographic attributes such as DOI, title, year, and cited-by count.
Node	`Affiliation`	Represents an institution and supports institution-aware filtering and analysis.
Relationship	`PUBLICATION`	Connects an Author to a Work to represent authorship.
Relationship	`AFFILIATED_WITH`	Connects an Author to an Affiliation to represent institutional membership.
Relationship	`CITED`	Connects a citing Work to a cited Work to represent citation structure.

Figure 4.3 illustrates how the UWB CSS faculty co-authorship and citation schema supports the identification of potential collaborators and possible faculty promotion referees. In this schema, a target UWB CSS faculty member is represented as an Author node connected to Work nodes through `PUBLICATION` relationships and to an Affiliation node through `AFFILIATED_WITH`. Other researchers can then be discovered through citation links when their published works cite the target faculty member's works. A researcher who frequently cites the target faculty member's publications may indicate strong topical alignment and awareness of the faculty member's research, making that researcher a potential collaborator. Furthermore, if such a researcher is not affiliated with the same institution as the target faculty member, has not co-authored with the target faculty member within the last five years, and is recognized as an established scholar in the relevant field, then that researcher may also represent a strong candidate to serve as an external faculty promotion referee. In this way, the schema is designed not only to model scholarly relationships, but also to support practical institution-centered exploration for collaboration and promotion-related evaluation.

4.4 Graph Loading on Neo4j, ArangoDB and MASS Graph DB

Figure 4.4 illustrates the common logical graph loading process used across all three platforms for the UWB-CSS-Citation dataset. In each case, loading follows the same staged workflow: initialize the graph environment, ingest author files, resolve author metadata and affiliations, ingest publication files, create fallback author records where necessary, and finally ingest cited-work files to establish citation relationships. This shared pipeline ensures that the same graph semantics are preserved across platforms, including the same vertex types (`Author`, `Work`, `Affiliation`), edge types (`PUBLICATION`, `AFFILIATED_WITH`, `CITED`), identity normalization rules, affiliation resolution logic, and deduplication behavior. The main differences are at the implementation and storage layer. Neo4j performs transactional loading into a persistent property graph using Cypher operations with constraints for identity management. ArangoDB Cloud implements the same logical process through graph collections, document-based vertex and edge insertion, and persistent indexes. MASS, in contrast, materializes the graph entirely in memory using its runtime graph structures,

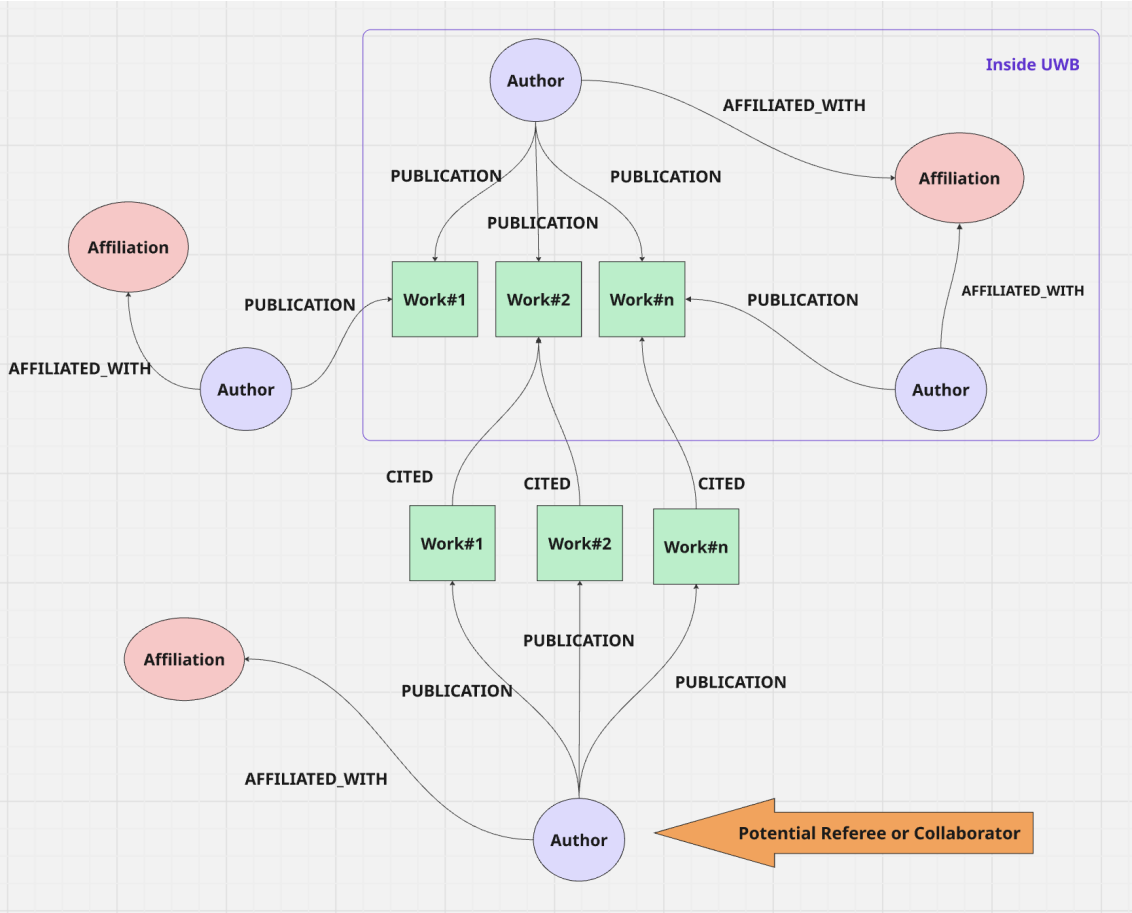


Figure 4.3: UWB CSS Faculty Co-authorship and Citation Schema

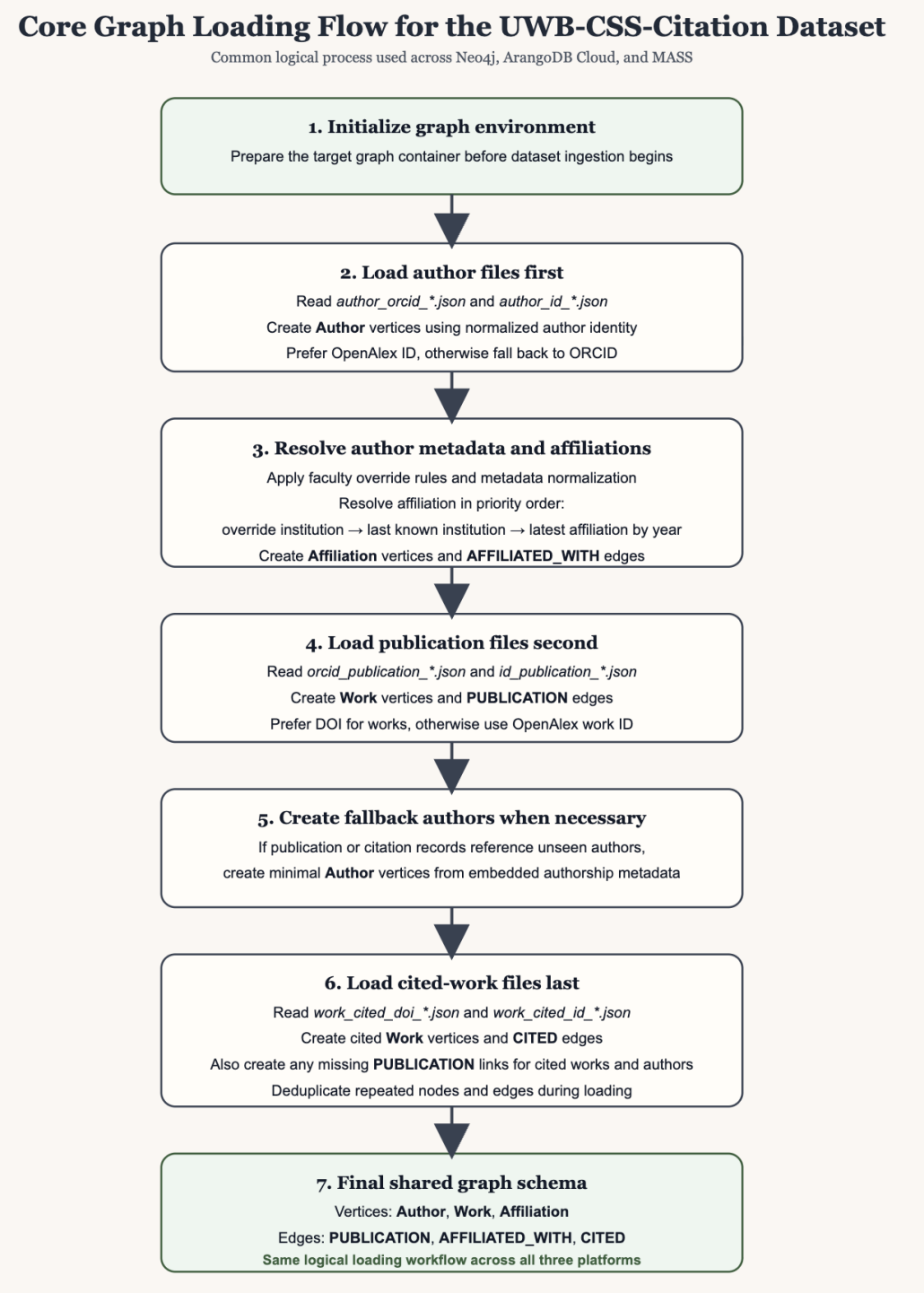


Figure 4.4: Graph loading workflow for the UWB-CSS-Citation dataset across Neo4j, ArangoDB Cloud, and MASS Graph DB.

meaning that the same logical graph is constructed without a persistent database backend. Thus, while the physical loading mechanisms differ, the underlying graph construction logic remains consistent across the three systems.

The construction process begins from the curated UWB CSS faculty seed list and expands through in-scope faculty publications, co-authors, affiliations, and citing works, preserving an institution-centered scholarly graph rather than an unbounded global crawl. During loading, author resolution prioritizes ORCID matching first and OpenAlex author ID matching second, while affiliation assignment follows a consistent fallback order using manual overrides, last-known institutions, and latest-year affiliation records when needed. Work nodes are created from both publication exports and cited-by exports, and PUBLICATION relationships connect authors to works while preserving authorship metadata such as author position and corresponding-author status. Finally, CITED relationships are inserted from citing works to cited works so that citation direction is represented consistently across Neo4j, ArangoDB, and MASS Graph DB.

The separation between shared logical parsing and backend-specific insertion allows the research to compare platform behavior while preserving the same underlying scholarly graph model.

4.5 Benchmarking Methodology and Evaluation Criteria

The benchmarking framework in this research is designed as an LDBC-inspired [23], workload-oriented evaluation for comparing graph database behavior across Neo4j, ArangoDB, and MASS Graph DB. Rather than measuring only one performance dimension, the framework evaluates write-path efficiency through bulk and incremental ingestion, read-path responsiveness through query throughput and average latency, bounded traversal behavior through fixed multi-hop traversal workloads, and graph completeness through node and relationship count validation. To support fair comparison, the benchmark uses the same workload categories, query types, time intervals, batch sizes, and iteration counts across the three platforms wherever possible, so that the resulting measurements reflect backend-specific execution behavior rather than differences in testing procedure.

The benchmark suite evaluates four major metric categories: bulk ingestion throughput,

incremental ingestion throughput, query throughput, and multi-hop traversal latency. These metrics are selected to capture both write-side and read-side behavior: ingestion metrics measure how efficiently each backend updates graph structure, while query and traversal metrics measure whether the graph remains responsive for interactive scholarly exploration.

1. **Bulk ingestion:** Bulk ingestion is defined as a high-intensity batched write workload in which the system inserts synthetic `PUBLICATION` edges in large batches. For Neo4j and ArangoDB, this metric reflects backend write performance under batched insertion, while for MASS Graph DB it should be interpreted as the cost of in-memory adjacency updates rather than full database-style transactional write throughput.

2. **Incremental ingestion:** Incremental ingestion is defined as a smaller, paced update workload that simulates online graph growth. Neo4j and ArangoDB insert smaller batches of synthetic `PUBLICATION` edges, while MASS Graph DB approximates the same workload through in-memory `PUBLICATION` edge insertion.

3. **Query throughput:** Query throughput is measured as sustained queries per second over a fixed time window. The benchmark uses a lightweight aggregation query based on counting author nodes as the primary read workload. Rather than retrieving full author-publication paths, each query performs a simple node-count aggregation over the author set, allowing Neo4j, ArangoDB, and MASS Graph DB to be compared using the same logical read pattern even though each system expresses the query in its own query language or API.

4. **Multi-hop traversal latency:** Multi-hop traversal latency is measured using bounded warm-cache traversal queries that follow co-authorship-style expansion semantics. This metric evaluates how efficiently each platform supports repeated neighborhood exploration, which is central to collaborator discovery and citation-based scholarly analysis.

Figure 4.5 presents the benchmarking workflow used to evaluate the UWB-CSS-Citation dataset. The process begins by preparing the benchmark environment and confirming that

the graph is available for testing, followed by execution of the main performance measurements: query throughput, multi-hop traversal latency, storage footprint, and ingestion performance. Individual test results are then aggregated into a final summary report. Although the three platforms differ in their underlying execution and storage models, this workflow provides a consistent evaluation structure for comparing their performance characteristics.

4.6 Data Validation and Correctness

Graph correctness is validated through both manual and automated checks to reduce errors introduced by incomplete metadata, ambiguous author identities, and backend-specific loading logic. The validation process first verifies faculty identities and publication records against trusted scholarly sources, then checks the loaded graph for consistent identifiers, required properties, duplicate entities, and correctly directed relationships.

1. **Faculty identity validation:** Each UWB CSS faculty member is manually checked for correct ORCID availability before being used as a seed author. When an ORCID is missing, incomplete, or not reliable enough for author lookup, the corresponding OpenAlex author ID is used as the alternative authoritative identifier to ensure that the correct faculty profile is collected.

2. **Publication completeness validation:** Each faculty member's publication list is manually compared against OpenAlex and external scholarly sources, including Google Scholar, ResearchGate, and faculty profile pages. When missing works are found and confirmed to belong to the faculty member, they are added to improve the completeness of the scholarly graph.

3. **Misplaced publication correction:** Incorrectly linked publications are manually reviewed and removed, especially when papers are associated with a faculty member because another researcher has the same or a similar name. This step reduces false author-work relationships and improves the reliability of later collaborator and referee-candidate analysis.

Figure 4.5: Benchmark process used across Neo4j, ArangoDB, and MASS Graph DB.

4. **Automated node and identifier checks:** Automated checks verify that ORCID, DOI, and OpenAlex identifiers are normalized and preserved after graph loading. These checks also confirm that duplicate `Author` and `Work` nodes are not created and that required node properties remain available for querying and benchmarking.

5. **Edge correctness validation:** Relationship validation checks whether `PUBLICATION` edges correctly connect authors to their works, `AFFILIATED_WITH` edges correctly represent institutional links, and `CITED` edges preserve the intended direction from citing work to cited work.

Chapter 5

GRAPH DATABASE PLATFORMS AND DATASETS FOR BENCHMARKING EVALUATION

This chapter presents the benchmark datasets, graph-density measures, platform execution environments, and backend-specific loading and testing mechanisms used to evaluate the UWB-CSS Citation Graph against Neo4j, ArangoDB, and MASS Graph DB under comparable cross-dataset benchmarking conditions.

5.1 Datasets for Benchmarking

This section introduces the datasets used to evaluate the performance behavior of the proposed heterogeneous scholarly citation graph. To support comparison across graphs with different sizes and structures, each dataset is described using its number of nodes, number of edges, and directed graph density. Directed graph density is defined as:

$$\text{Density} = \frac{E}{N(N-1)}$$

where N represents the number of graph nodes and E represents the number of directed edges. This metric is used to contextualize benchmark results because graph density affects how many relationships may be encountered during ingestion, query execution, and multi-hop traversal.

Table 5.1 summarizes the datasets used to contextualize the benchmark results of the UWB-CSS Citation Graph. These datasets were mostly selected from the Stanford SNAP [24] dataset collection to provide external comparison graphs with different topology, size, and density characteristics. The UWB-CSS Citation Graph serves as the target heterogeneous scholarly graph, while cit-HepTh is used as a domain-relevant single-relation citation baseline for evaluating whether the proposed graph is qualitatively and quantitatively competitive with a simpler citation-only network. The p2p-Gnutella08 and wiki-Vote datasets

provide additional directed graph comparisons: p2p-Gnutella08 represents a smaller peer-to-peer topology with hub-like behavior, while wiki-Vote represents a denser interaction network. LDBC SNB Interactive SF3 is included as the standardized benchmark baseline; in this research, it is loaded from the `interactive-updates-sf3/inserts/` directory of the LDBC SNB SF3 [23] workload as a self-contained subgraph containing `Person`, `Forum`, `Post`, and `Comment` nodes with `KNOWS`, `LIKES`, and `HAS_MEMBER` relationships [23]. Together, these datasets allow the later results to evaluate whether the ingestion throughput, query throughput, and multi-hop traversal behavior of the UWB-CSS Citation Graph follow reasonable trends relative to graph structure, rather than being interpreted in isolation.

Figure 5.1 visualizes the benchmark datasets in Table 5.1 as a conceptual topology spectrum based on schema heterogeneity and directed graph density. LDBC SNB Interactive SF3 has the largest node count among the evaluated datasets and is included as a heterogeneous benchmark baseline because it contains multiple node types, including `Person`, `Forum`, `Post`, and `Comment`, and multiple relationship types, including `KNOWS`, `LIKES`, and `HAS_MEMBER`. The UWB-CSS Citation Graph is also heterogeneous because it models `Author`, `Work`, and `Affiliation` nodes together with `PUBLICATION`, `AFFILIATED_WITH`, and `CITED` relationships; however, unlike LDBC, it is institution-centered and designed around scholarly citation, authorship, and affiliation analysis. In contrast, cit-HepTh, p2p-Gnutella08, and wiki-Vote are treated as single-relation datasets because each primarily uses one node type and one dominant edge type: Paper-to-Paper citation for cit-HepTh, Peer-to-Peer connection for p2p-Gnutella08, and WikiUser-to-WikiUser voting for wiki-Vote. cit-HepTh is especially useful as a comparison dataset because it has a similar node count to the UWB-CSS Citation Graph but a much denser citation-only structure, allowing the evaluation to separate the effect of graph density from the effect of schema heterogeneity. p2p-Gnutella08 is smaller but represents a sparse peer-to-peer topology with hub-like behavior, where some peers may have disproportionately more connections than others. wiki-Vote has the highest density among the selected datasets, making it useful for observing how denser single-relation interaction graphs affect ingestion, query throughput, and traversal behavior. Together, these datasets allow the UWB-CSS Citation Graph to be evaluated not in isolation, but against graphs that vary by size, density, relation type, and topology.

Table 5.1: Dataset Characteristics Used for Cross-Topology Benchmarking

Dataset	**Description**	**Node / Edge Types**	**Nodes**	**Edges**	**Density**
UWB-CSS Citation Graph	Heterogeneous scholarly citation graph	Nodes: Author, Work, Affiliation; Edges: PUBLICATION, AFFILI-ATED_WITH, CITED	29,557	34,801	0.00003984
LDBC SNB Interactive SF3	Standard social-network benchmark baseline	Nodes: Person, Forum, Post, Comment; Edges: KNOWS, LIKES, HAS_MEMBER	713,714	25,143	0.0000000494
cit-HepTh	ArXiv high-energy physics citation network	Nodes: Paper; Edges: CITES	27,770	352,807	0.00045751
p2p-Gnutella08	Peer-to-peer file sharing network	Nodes: Peer; Edges: CONNECTS_TO	6,301	20,777	0.00052340
wiki-Vote	Wikipedia voting interaction network	Nodes: WikiUser; Edges: VOTED_FOR	7,115	103,689	0.00204854

The UWB-CSS Citation Graph is sparse mainly because of its institution-centered crawl scope. The graph includes UWB CSS faculty, their publications, co-authors, citing papers, and citing-paper authors as context nodes, but it does not recursively expand all publications, citations, and neighboring authors for every external context node. Expanding all of those relationships would make the graph much denser, but it would also introduce many unrelated nodes and weaken the focus on faculty-centered collaborator and referee-candidate exploration. Therefore, the graph's sparsity is an expected design outcome.

To maintain an approximately apple-to-apple comparison across datasets, the benchmark keeps the same workload categories and execution settings wherever possible. Bulk ingestion, query throughput, and bounded 3-hop traversal latency are evaluated consistently across all datasets. For throughput, each dataset uses a lightweight node-count aggregation query as the primary read workload. For traversal latency, each dataset uses a bounded 3-hop query with equivalent traversal depth, while the relationship pattern is adapted to the dataset schema, such as `Author/PUBLICATION` for UWB-CSS, `Paper/CITES` for cit-HepTh, `Peer/CONNECTS_TO` for p2p-Gnutella08, `WikiUser/VOTED_FOR` for wiki-Vote, and `Person/KNOWS` for LDBC. The measurement settings are also kept consistent, including fixed test duration, concurrency, hop count, and repeated runs. This preserves fairness at the workload level while still respecting each dataset's native graph structure.

5.2 Neo4j

Table 5.2 summarizes the execution environment used for Neo4j graph loading and benchmarking. The table reports both the hardware and operating-system context and the software context in order to document the experimental setup used for this platform. Specifically, it shows that Neo4j benchmarking was conducted in a Linux-based environment and accessed through Python using the official `neo4j` driver over the Bolt protocol. Providing this information helps clarify the runtime conditions under which the Neo4j ingestion and query measurements were obtained and supports reproducibility of the benchmark configuration.

The Neo4j evaluation follows the same logical benchmark categories as the other platforms, but its implementation and mechanism differ from the following perspectives:

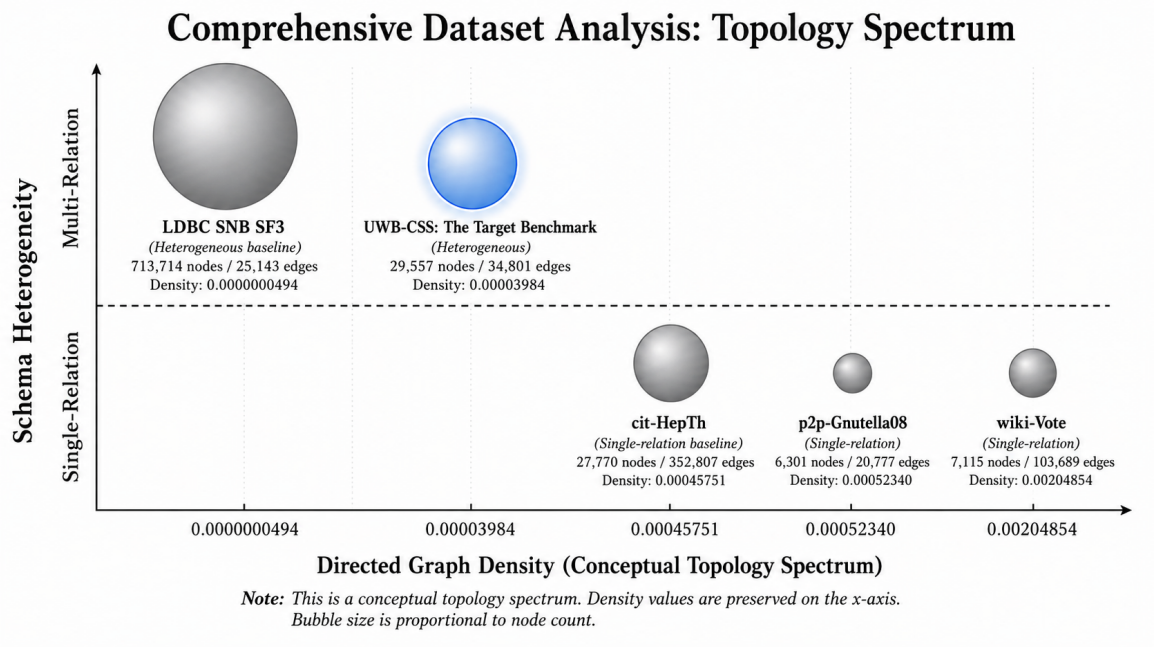


Figure 5.1: Conceptual Dataset Topology Spectrum

1. **Neo4j graph loading process:** Neo4j loads the UWB-CSS Citation Graph through an incremental, transaction-driven process that differs from the two-phase document insertion used in ArangoDB and the in-memory graph construction used in MASS Graph DB. During loading, each crawled OpenAlex JSON file is parsed and immediately translated into Cypher operations that `MERGE` or create the corresponding nodes and relationships. Schema constraints are used to preserve identifier uniqueness for authors, works, and affiliations, helping the loader avoid duplicate entities while supporting repeatable graph construction.

2. **Bulk ingestion testing:** Neo4j bulk ingestion is tested as a batched write workload that inserts synthetic authorship-style relationships into the existing graph using Cypher-based batch operations. The test groups many insertions into larger batches and measures the number of inserted graph elements per second over the benchmark interval. This metric reflects Neo4j's write-path behavior under high-volume trans-

actional insertion, including the cost of Cypher execution, constraint checks, and persistent storage updates.

3. **Query throughput testing:** Neo4j query throughput is measured by repeatedly executing the same logical node-count aggregation query over a fixed time window and reporting the completed queries per second. This test evaluates Neo4j's read-path responsiveness for a common scholarly exploration pattern, where the database retrieves a limited set of authors and their connected works through the authorship relationship. Because Neo4j uses Cypher for this workload, the result reflects the performance of its query planner, execution engine, indexing strategy, and graph traversal model under repeated read queries.

4. **Multi-hop traversal testing:** The multi-hop traversal workload is logically shared across all three platforms, but Neo4j implements it through Cypher pattern-matching queries over its native property-graph storage model. The test measures bounded traversal latency, especially the 3-hop workload, by expanding from authors through connected works and related authors or citation-linked works. In Neo4j, this result reflects the performance of Cypher traversal execution, indexed starting-point lookup, and native relationship navigation under an interactive scholarly exploration pattern.

Table 5.2: Neo4j execution environment used for graph loading and benchmarking.

Hardware / OS context	**Software context**
Linux-5.14.0-611.34.1.el9_7.x86_64-x86_64-with-glibc2.34; system Linux; processor x86_64	Python 3.9.25; Neo4j accessed through the Python `neo4j` driver via Bolt at `bolt://hermes25.uwb.edu:7687`

5.3 ArangoDB

Table 5.3 presents the execution environment used for ArangoDB graph loading and benchmarking. It includes the local hardware and operating-system context together with the software access configuration used to connect to the ArangoDB Cloud deployment. In particular, the table indicates that the experiments were initiated from a macOS-based client environment through `python-arango` and executed against the specified ArangoDB Cloud endpoint and database instance. This setup description is important because it distinguishes the client-side execution context from the remote database service used in the benchmark.

The ArangoDB evaluation follows the same logical benchmark categories as the other platforms, but its implementation and mechanism differ from the following perspectives:

1. **ArangoDB graph loading process:** ArangoDB loads the UWB-CSS Citation Graph through a two-phase bulk-ingestion process that differs from Neo4j's transaction-driven Cypher loading and MASS Graph DB's in-memory Java construction. In the first phase, the loader parses the crawled OpenAlex JSON files, collects vertex and edge records in memory, and deduplicates them before insertion. In the second phase, the cleaned records are inserted into ArangoDB collections, where vertices are stored as documents with safe `_key` values and relationships are stored as edge documents using `_from` and `_to` references. This process preserves the same logical graph structure while adapting it to ArangoDB's document-oriented graph model.

2. **Bulk ingestion testing:** ArangoDB bulk ingestion is tested as a batched document and edge insertion workload. The benchmark inserts synthetic authorship-style edge documents into the corresponding edge collection and measures the number of inserted elements per second over the benchmark interval. This metric reflects ArangoDB's write-path behavior under collection-based insertion, including document creation, edge-reference validation, index maintenance, and coordination overhead when the database is deployed in a cloud or clustered environment.

3. **Query throughput testing:** The query throughput workload is logically shared

across all three platforms, but ArangoDB implements it through AQL over its document and graph collections. The benchmark repeatedly executes an equivalent node-count aggregation query over a fixed time window, retrieving a limited set of authors and their connected works through the authorship relationship. This result reflects ArangoDB's read-path responsiveness under repeated AQL execution, including collection access, edge traversal, and query coordination through the ArangoDB client and server.

4. **Multi-hop traversal testing:** The multi-hop traversal workload is also logically shared across all three platforms, but ArangoDB executes it using AQL traversal syntax over named graph collections or edge collections. The benchmark measures bounded traversal latency, especially the 3-hop workload, using the same co-authorship-style expansion semantics applied in the other backends. In ArangoDB, this result should be interpreted through the cost of AQL traversal execution, edge-collection access, and possible cross-shard or coordinator overhead in distributed deployments.

Table 5.3: ArangoDB execution environment used for graph loading and benchmarking.

Hardware / OS context	**Software context**
macOS-26.3-arm64-arm-64bit; Darwin; arm processor; Python 3.9.6	ArangoDB Cloud via `python-arango`; endpoint `https://dd3fdaaf11d7.arangodb.cloud:8529`; database `uwbcss_citation_db`.

5.4 MASS Graph DB

Table 5.4 summarizes the execution environment used for MASS Graph DB graph loading and benchmarking. The table records the Linux server environment, the Java runtime context, and the MASS-specific execution configuration used in the experiments. In particular, it shows that MASS was executed through `mass-core.jar` using the `PropertyGraphPlaces`

Java API, with benchmark node settings defined in `nodes.xml`. This information is included to make the MASS runtime environment explicit and to document the system-level conditions under which its loading and benchmarking results were produced.

The MASS Graph DB evaluation follows the same logical benchmark categories as the other platforms, but its implementation and mechanism differ from the following perspectives:

1. **MASS Graph DB loading process:** MASS Graph DB loads the UWB-CSS Citation Graph through an in-memory, API-based construction process that differs from Neo4j's Cypher-based transactional loading and ArangoDB's document-collection insertion model. During loading, the OpenAlex JSON files are parsed by the Java loader, and vertices and edges are created directly through MASS_Java_Core API calls. Identity resolution and deduplication are managed through internal maps and caches, which track existing authors, works, affiliations, and relationships before new graph elements are inserted into the in-memory graph structure.

2. **Bulk ingestion testing:** MASS bulk ingestion is measured as the time required to construct or update the in-memory graph structure through direct vertex and edge insertion. Unlike Neo4j and ArangoDB, this measurement does not represent full transactional database write throughput or persistent storage commits. Instead, it reflects the cost of parsing graph records, resolving identifiers, and updating in-memory adjacency structures inside MASS Graph DB.

3. **Query throughput testing:** The query throughput workload is logically shared across all three platforms, but MASS Graph DB executes it through Java-based graph access and agent-style traversal over its in-memory graph representation. The benchmark repeatedly runs an equivalent node-count aggregation query over a fixed time window and reports completed queries per second. This result reflects MASS Graph DB's read-path behavior under in-memory execution rather than server-based query processing through Cypher or AQL.

4. **Multi-hop traversal testing:** The multi-hop traversal workload is also logically shared across the three platforms, but MASS Graph DB implements it through in-memory graph traversal and agent-based execution over MASS graph structures. The benchmark measures bounded traversal latency, especially the 3-hop workload, using the same co-authorship-style expansion semantics used for Neo4j and ArangoDB. In MASS, this result should be interpreted through the cost of agent dispatch, graph traversal, synchronization, and memory access across the distributed runtime.

Table 5.4: MASS execution environment used for graph loading and benchmarking.

Hardware / OS context	**Software context**
Linux `hermes2.uwb.edu` 5.14.0-611.13.1.el9_7.x86_64 #1 SMP PREEMPT_DYNAMIC Fri Dec 12 11:55:11 UTC 2025; x86_64 x86_64 x86_64 GNU/Linux	OpenJDK 17.0.18 (Temurin); MASS executed through `mass-core.jar` using the MASS `PropertyGraphPlaces` Java API; benchmark node configuration defined in `nodes.xml`.

Chapter 6

RESULTS

This chapter presents the evaluation results of the UWB-CSS Citation Graph, including visual verification of graph correctness, baseline platform performance, cross-dataset comparisons, comparison with a single-relation citation graph, and platform-specific benchmarking results across Neo4j, ArangoDB, and MASS Graph DB.

6.1 DB and System Verification through Visualization

The following visualizations verify that the graph database correctly connects authors, works, affiliations, and citation links across multiple relationship types and demonstrates that the heterogeneous scholarly graph can support exploratory, multi-hop analysis around a single faculty member.

The Cypher query in Figure 6.1 matches the target author by ORCID and optionally expands to affiliations, publications, citation-linked works, co-authors, and citing authors. The use of optional matches allows partial subgraphs to be returned, supporting both data validation and graph structure inspection. This figure also shows a subgraph produced from an author-seeded query (Professor Annuska Zolyomi, ORCID 0000-0002-9317-1786), including her affiliation, publications, co-author neighborhoods, and inbound/outbound citation links. Blue nodes denote researchers (Author), red nodes denote institutions (Affiliation), and yellow nodes denote papers (Work). Grey edges represent the focal author's publication relationships (`:PUBLICATION`), while orange edges indicate inbound citation relationships (`:CITED`) where a paper cites the given work. Overall, the network highlights Christian Reuter, shown as the blue node to the right of Annuska Zolyomi's author node at the bottom of the image, as a prominent citing author who has written multiple papers citing Zolyomi's work. This suggests that he could be a potential collaborator or a strong preliminary referee candidate for faculty promotion.

```
MATCH (a:Author {orcid: "0000-0002-9317-1786"})
OPTIONAL MATCH (a)-[rAff:AFFILIATED_WITH]->(aff:Affiliation)
OPTIONAL MATCH (a)-[rPub:PUBLICATION]->(w:Work)
// works that w cites OR are cited by w
OPTIONAL MATCH (w)-[rC:CITED]-(otherWork:Work)
// co-authors on this author's own works
OPTIONAL MATCH (w)<-[rCoPub1:PUBLICATION]-(coAuthor1:Author)
WHERE coAuthor1 <> a

// co-authors on works connected by citation
OPTIONAL MATCH (otherWork)<-[rCoPub2:PUBLICATION]-(coAuthor2:Author)
WHERE coAuthor2 <> a

// authors who cited this author's work
OPTIONAL MATCH (citingWork:Work)-[rCitedHis:CITED]->(w)
OPTIONAL MATCH (citingAuthor:Author)-[rCitingPub:PUBLICATION]->(citingWork)
WHERE citingAuthor <> a
RETURN a,
       rAff, aff,
       rPub, w,
       rC, otherWork,
       rCoPub1, coAuthor1,
       rCoPub2, coAuthor2,
       rCitedHis, citingWork,
       rCitingPub, citingAuthor;
```

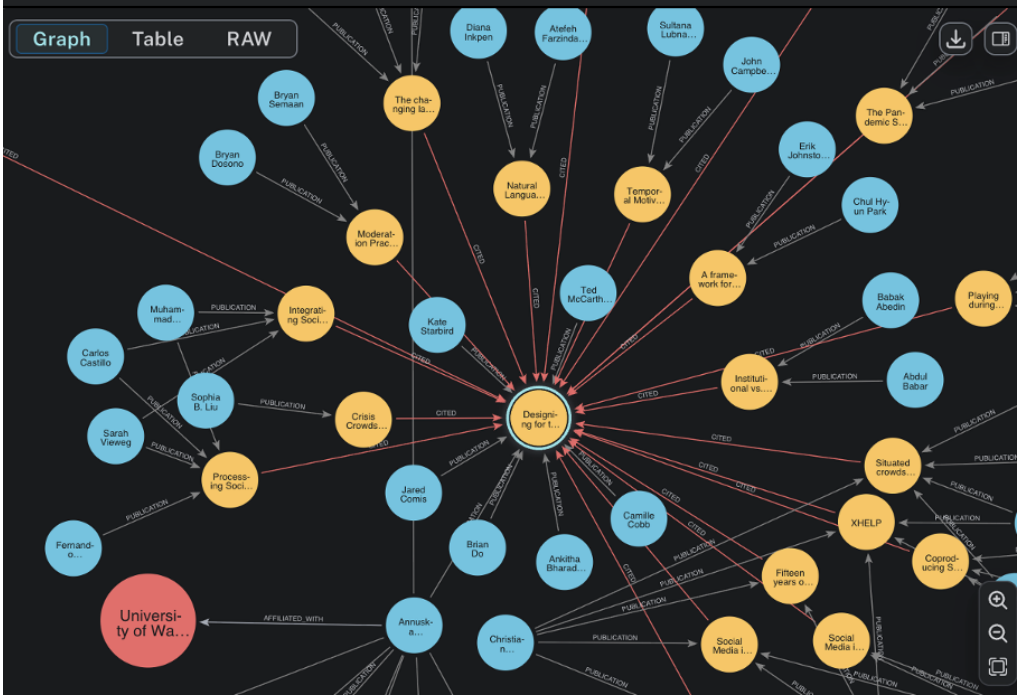


Figure 6.1: Emerging Clusters Around a Single Author: Professor Zolyomi's Co-Authorship and Citation Network.

Figure 6.2 shows a citation-frequency-based Cypher query used to identify external scholars related to the target UWB CSS faculty member. The query starts from the target author's publications, finds works that cite those publications, and retrieves the authors of the citing works. It then excludes the target author, filters out University of Washington-affiliated authors, and removes authors who have co-authored with the target author within the last five years. The returned candidates are ranked by the number of distinct citing works, with the top results shown in the figure. Therefore, the query supports both potential collaborator discovery and preliminary faculty promotion referee identification by highlighting external researchers who are familiar with the target author's work while reducing institutional and recent-collaboration conflicts. The returned authors, with only the top three candidates shown, are ranked by citation frequency and their ORCID information is included to support further candidate evaluation.

6.2 Baseline Platform Performance Using the LDBC Dataset

The local laptop benchmarking test results in Table 6.1 show clear workload-specific strengths across the three graph systems. MASS Graph DB achieves the highest bulk ingestion throughput, suggesting that its in-memory design is well suited for graph-loading workloads. Neo4j performs best for interactive graph querying, with the highest query throughput and the lowest 3-hop traversal latency, indicating strong support for latency-sensitive traversal operations. ArangoDB performs between Neo4j and MASS in most categories, showing moderate ingestion and query performance. Overall, this table suggests that MASS benefits ingestion-heavy workloads, while Neo4j is strongest for user-facing graph traversal and query responsiveness.

The actual deployment platform baseline results in Table 6.2 show a different but still workload-dependent pattern. MASS Graph DB on the hermes1-24.uwb.edu cluster system continues to provide the highest bulk ingestion throughput, confirming its strength in graph loading even in the testing environment. ArangoDB Cloud achieves the highest query throughput among the three deployed systems, suggesting that its managed cloud configuration can support concurrent query execution effectively. Neo4j on the hermes25.uwb.edu provides the lowest 3-hop traversal latency, showing that it remains the strongest system

```
MATCH (a:Author {orcid: "0000-0002-9317-1786"})-[:PUBLICATION]->(aWork:Work)
MATCH (citingWork:Work)-[:CITED]->(aWork)
MATCH (author:Author)-[:PUBLICATION]->(citingWork)
WHERE author.orcid <> "0000-0001-7285-2569"
  // exclude UWB-affiliated authors
  AND NOT EXISTS {
    MATCH (author)-[:AFFILIATED_WITH]->(uwbAff:Affiliation)
    WHERE toLower(uwbAff.name) CONTAINS "university of washington"
  }
  // exclude authors who co-authored with the target author in the last 5 years
  AND NOT EXISTS {
    MATCH (author)-[:PUBLICATION]->(coWork:Work)<-[:PUBLICATION]-(a)
    WHERE coWork.publication_year >= date().year - 5
  }
OPTIONAL MATCH (author)-[:AFFILIATED_WITH]->(aff:Affiliation)

RETURN author.name AS author,
       author.orcid AS orcid,
       COUNT(DISTINCT citingWork) AS citing_frequency
ORDER BY citing_frequency DESC, author ASC;
```

Table | RAW

	author	orcid	citing_frequency
1	"Christian Reuter"	"0000-0003-1920-038X"	5
2	"Marc-André Kaufhold"	"0000-0002-0387-9597"	4
3	"Anke Brock"	"0000-0002-0017-396X"	3

Figure 6.2: Potential Faculty Promotion Referee Discovery with Affiliation and Recent Co-authorship Filtering: An example from Professor Zolyomi

for low-latency multi-hop graph exploration. This table supports the conclusion that no single system dominates all workloads: MASS benefits ingestion, ArangoDB Cloud benefits query throughput, and Neo4j benefits traversal latency.

Comparing the local results with the actual testing environment results shows that deployment conditions have a major impact on benchmark performance. All three systems show lower performance in the actual testing environment for at least some metrics, especially query throughput and traversal latency. This suggests that hardware differences, cloud-network overhead, runtime configuration, dataset loading method, and system-specific deployment constraints can significantly affect measured results. Therefore, the results should not be interpreted as absolute rankings of the database engines alone, but as workload-specific performance patterns under specific deployment settings. This comparison also justifies presenting the two tables separately, because it separates general platform tendencies from the effects of the final benchmark environment.

Table 6.1: LDBC Baseline Performance on Local Platform Settings

Metric	**Neo4j**	**ArangoDB**	**MASS Graph DB**
Bulk Ingestion Throughput (element/s)	14,584	12,307	79,430
Query Throughput (QPS)	4,228.50	1,233.42	0.80
3-Hop Traversal Avg Latency (ms)	0.85	18.92	1314.94

6.3 Performance Comparison Across Datasets

The bulk ingestion results in Figure 6.3 show that ingestion performance is influenced by graph size, density, degree distribution, and schema heterogeneity. In general, larger graphs increase materialization cost, denser graphs require more relationship insertions, high-degree hubs may concentrate edge writes around a small set of nodes, and heterogeneous schemas add overhead because multiple node and relationship types must be resolved during inges-

Table 6.2: LDBC Baseline Performance on Test Deployment Platforms

Metric	**Neo4j on hermes25.uwb.edu**	**ArangoDB Cloud**	**MASS Graph DB on the hermes1-24.uwb.edu cluster system**
Bulk Ingestion Throughput (elements/sec)	8,882.97	3,877.54	28,683.45
Query Throughput (QPS)	20.82	91.98	0.12
3-Hop Traversal Avg Latency (ms)	2.33	105.58	10,398.48

tion. On Neo4j, the UWB-CSS Citation Graph reaches 9,396.74 edges/sec, close to the heterogeneous LDBC SNB SF3 baseline at 8,882.97 elements/sec, but lower than simpler single-relation datasets such as cit-HepTh and wiki-Vote. This suggests that schema heterogeneity may introduce additional ingestion cost compared with single-relation graphs, while p2p-Gnutella08 performs lower despite its smaller size, likely due to hub-like peer-to-peer topology. On ArangoDB, the UWB-CSS graph also performs close to LDBC SNB SF3, again indicating that heterogeneous graph loading and edge-collection management can reduce ingestion throughput. On MASS Graph DB, UWB-CSS remains close to the single-relation datasets, while LDBC SNB SF3 drops sharply, likely because its much larger node count creates greater memory-management overhead in the current in-memory MASS implementation. Overall, the UWB-CSS citation bulk ingestion result is consistent with the expected behavior of a medium-scale heterogeneous scholarly graph: its multiple node and relationship types appear to add ingestion overhead, while platform-specific execution models determine how strongly that overhead affects throughput.

The query throughput results in Figure 6.4 show how graph size and schema heterogeneity affect lightweight read workloads. In this benchmark, the query is a simple node-count aggregation over one selected node type, so the result reflects how efficiently each platform

can access and aggregate a target collection of nodes rather than execute complex multi-hop traversal. On Neo4j, the UWB-CSS Citation Graph reaches 497.09 QPS, lower than the simpler single-relation datasets p2p-Gnutella08, wiki-Vote, and cit-HepTh, but much higher than the larger heterogeneous LDBC SNB SF3 dataset at 20.82 QPS. This suggests that both heterogeneity and graph size contribute to query overhead in Neo4j: the UWB-CSS graph is more complex than single-relation graphs, but much smaller than LDBC, so its throughput remains in the middle range. On ArangoDB, all datasets produce relatively close throughput values, which is consistent with its document-based storage model, where lightweight count-style aggregation over collections can be handled efficiently and is less sensitive to graph topology than traversal-heavy workloads. On MASS Graph DB, the UWB-CSS graph reaches 0.1693 QPS, higher than cit-HepTh and LDBC SNB SF3 and close to wiki-Vote, showing that its lightweight query behavior remains within the expected range for the current in-memory MASS implementation. Overall, this result supports the validity of the UWB-CSS Citation Graph because its lightweight query performance follows the expected behavior of a medium-scale heterogeneous graph: slower than simpler single-relation datasets on Neo4j, but substantially more efficient than a much larger heterogeneous workload.

The 3-hop traversal latency results in Figure 6.5 show that traversal performance is affected by density, degree distribution, schema heterogeneity, and dataset size. In general, denser graphs or graphs with high-degree hubs can create more candidate paths during multi-hop expansion, increasing latency. This pattern is visible on Neo4j, where p2p-Gnutella08 and wiki-Vote show higher latency than the sparser cit-HepTh and LDBC SNB SF3 datasets. The UWB-CSS Citation Graph reaches 7.38 ms on Neo4j, which is higher than cit-HepTh and LDBC but lower than p2p-Gnutella08 and wiki-Vote. This result is explainable because the UWB-CSS traversal cannot be expressed as a simple homogeneous 3-hop path over the same node type and relationship type. Even though LDBC is a heterogeneous dataset overall, the traversal workload used for LDBC can follow a direct person-to-person pattern through repeated `KNOWS` relationships. Similarly, single-relation datasets such as cit-HepTh, p2p-Gnutella08, and wiki-Vote can follow repeated relationships between the same type of node. In contrast, the UWB-CSS co-authorship traversal

must alternate between Author and Work nodes through authorship relationships in order to discover connected scholars. This alternating traversal requires the database to resolve different node types and relationship directions, which introduces additional schema-related cost. However, the relative sparsity of the UWB-CSS graph limits the number of candidate paths and partially offsets the overhead introduced by this alternating-node traversal. A similar effect appears on ArangoDB, where UWB-CSS has the highest 3-hop latency because its traversal crosses heterogeneous node types rather than following one repeated edge pattern. For MASS Graph DB, dataset size strongly affects traversal: LDBC SNB SF3 is the slowest because its large node count creates heavy memory-traversal overhead, while UWB-CSS is much faster than p2p-Gnutella08, suggesting that its sparse structure helps reduce traversal expansion despite its heterogeneous schema. Overall, the UWB-CSS result is valid for its topology because its latency reflects the expected cost of a richer heterogeneous scholarly traversal, while its sparsity prevents the traversal latency from becoming an extreme outlier.

Overall, the UWB-CSS Citation Graph is valid as a benchmark dataset because its performance follows the expected relationship between graph topology, size, density, and system behavior. It is not expected to outperform all single-relation datasets because it models a richer scholarly structure with multiple node and edge types. Instead, the key validation point is whether its results are explainable relative to comparable datasets. For bulk ingestion, the UWB-CSS graph is close to LDBC SNB SF3 on Neo4j and ArangoDB and performs strongly on MASS. For lightweight query throughput, the UWB-CSS graph falls between simpler single-relation datasets and the much larger heterogeneous LDBC workload on Neo4j and ArangoDB, which is consistent with its medium scale and heterogeneous schema. For 3-hop traversal, the UWB-CSS graph shows moderate latency on Neo4j and higher but explainable latency on ArangoDB because its traversal requires alternating between author and work nodes rather than repeatedly following one homogeneous relationship type. Therefore, across the three metrics, the UWB-CSS Citation Graph does not appear as an anomalous dataset; its performance is explainable based on its topology and supports its use as a practical heterogeneous scholarly benchmark.

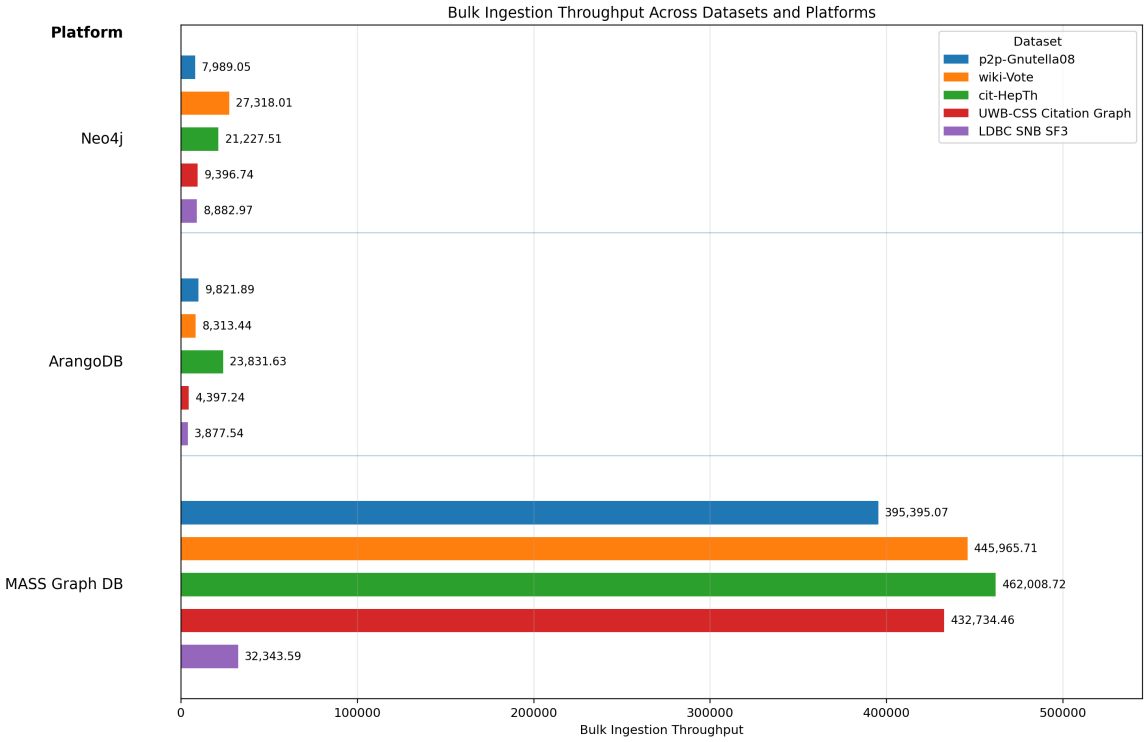


Figure 6.3: Bulk Ingestion Throughput Across Dataset Topologies on Neo4j, ArangoDB, and MASS Graph DB

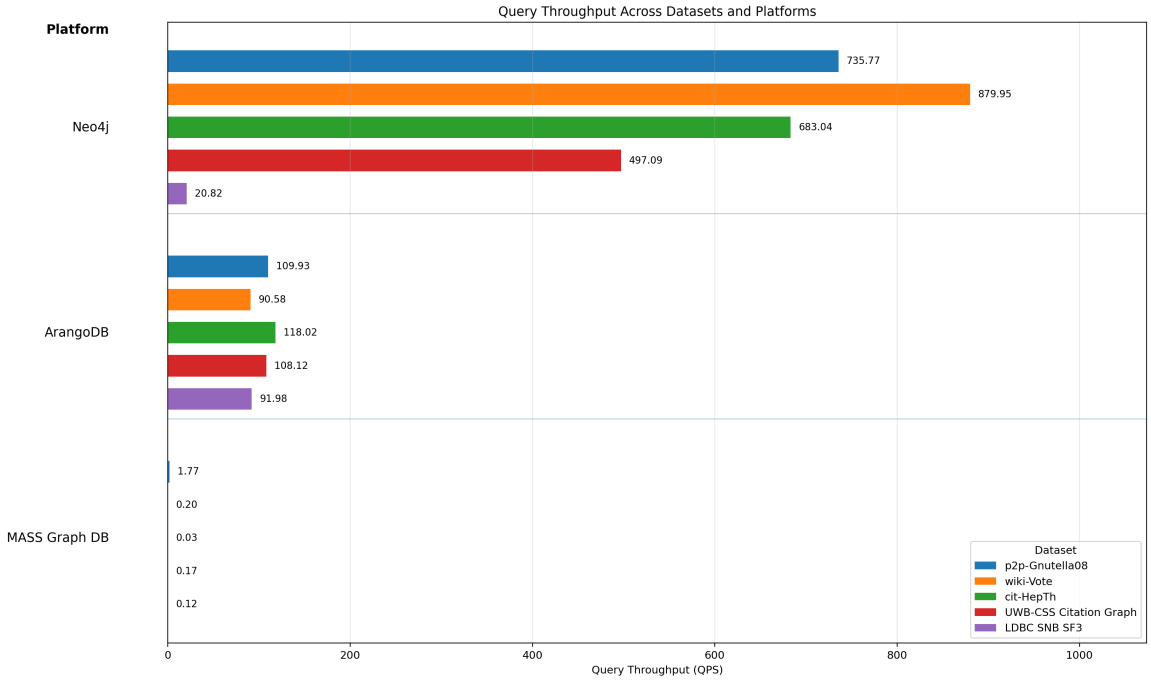


Figure 6.4: Query Throughput Across Dataset Topologies on Neo4j, ArangoDB, and MASS Graph DB

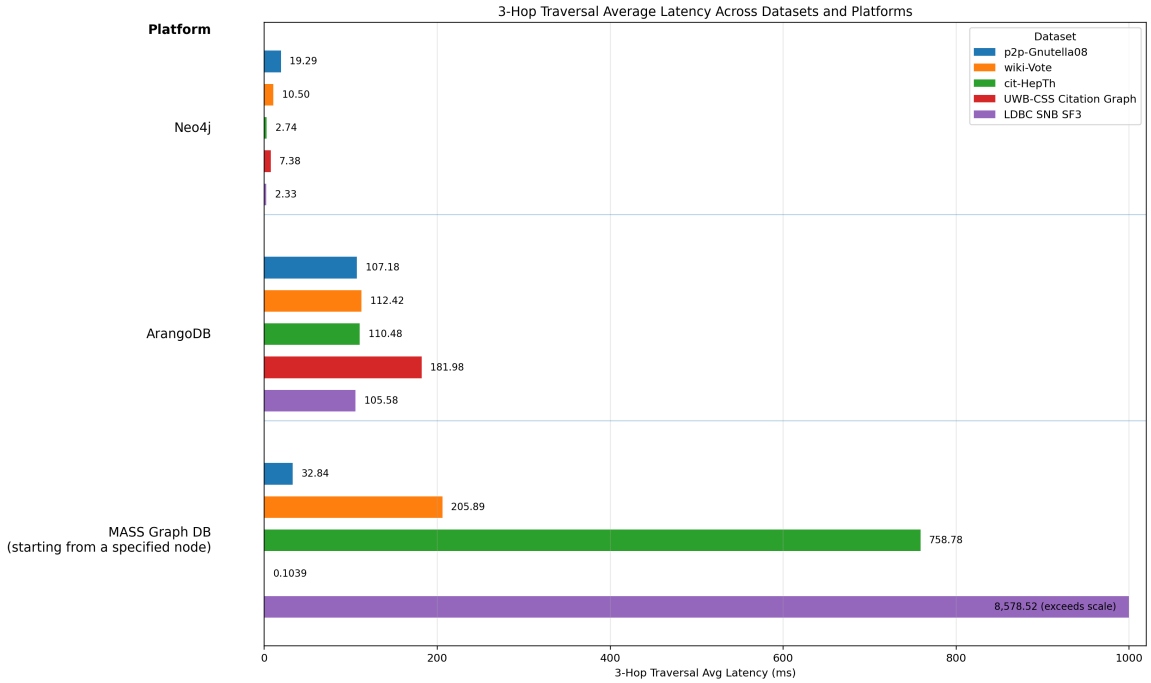


Figure 6.5: 3-Hop Traversal Latency Across Dataset Topologies on Neo4j, ArangoDB, and MASS Graph DB

6.4 Comparison Between the UWB-CSS Citation Graph and a Single-Relation Citation Graph on Neo4j, ArangoDB and MASS Graph DB

As can be seen in Table 6.3, the Neo4j comparison results are consistent with the expected trade-off between graph simplicity and analytical expressiveness. The single-relation citation graph achieves stronger raw ingestion and traversal performance because it models only paper-to-paper citation relationships. In contrast, the UWB-CSS Citation graph introduces a heterogeneous schema with authors, works, affiliations, publication edges, affiliation edges, and citation edges, which increases ingestion and traversal overhead. However, the query throughput decreases by only 27.22%, and the average 3-hop traversal latency remains low at 7.38 ms, suggesting that the richer graph design preserves practical interactive query performance while enabling more expressive scholarly analysis, such as collaborator discovery and potential referee identification.

Table 6.3: Neo4j Single-Server Performance Comparison: UWB-CSS-Citation Graph vs. Single-Relation Citation Graph

Metric	**Single-Relation Citation Graph**	**UWB-CSS-Citation Graph**	**Difference**
Total Nodes	27,770	29,557	+6.44%
Total Relationships	352,807	34,801	-90.14%
Bulk Ingestion Throughput (elements/sec)	21,227.51	9,396.74	-55.73%
Incremental Ingestion Throughput (edges/sec)	11,702.75	843.62	-92.79%
Query Throughput (QPS)	683.04	497.09	-27.22%
Query Avg Latency (ms)	14.60	20.08	+37.53%
3-Hop Traversal Avg Latency (ms)	2.74	7.38	+169.34%

Figure 6.6 shows that the UWB-CSS Citation Graph and the single-relation citation

graph respond differently to multi-node ArangoDB Cloud scaling. For bulk and incremental ingestion, the UWB-CSS graph shows a small but visible recovery from 4 nodes to 8 nodes, suggesting that once the cluster is large enough, its heterogeneous structure will benefit from parallel distribution of different node and edge collections. In contrast, the single-relation citation graph continues to decline as the number of nodes increases, indicating that for a more homogeneous graph, the added coordination and distribution overhead will likely outweigh the benefit of parallel ingestion. A similar pattern appears more strongly in the 3-hop traversal latency results. The UWB-CSS graph has higher traversal latency overall because its heterogeneous traversal must cross different node types and relationship directions, but its latency decreases from 4 nodes to 8 nodes, suggesting that additional nodes can partially reduce the cost of distributed heterogeneous traversal. The single-relation citation graph does not show the same improvement and instead becomes slower as the cluster grows, likely because its simpler traversal gains less from distribution while still paying cross-node coordination cost. The lightweight query latency shows a slightly different trend because this workload mainly counts or aggregates nodes rather than traversing graph structure. For this type of query, additional distribution does not necessarily improve performance, since the system may need to coordinate partial aggregation results across shards. Overall, the figure suggests that the UWB-CSS graph introduces higher execution cost due to heterogeneity, but it also has stronger potential to benefit from multi-node parallelism for ingestion and traversal once the cluster size is sufficient.

The MASS Graph DB bulk-ingestion results in Figure 6.7 show that the UWB-CSS Citation Graph and the single-relation citation graph follow a similar performance pattern across the multi-node configurations. This comparison is meaningful because the two graphs have a relatively close number of nodes, while differing in edge volume, density, and schema design: the single-relation citation graph is denser and has more edges, whereas the UWB-CSS Citation Graph is sparser and structurally more heterogeneous. The sharp decrease in bulk-ingestion throughput under multi-node execution is mainly caused by the current distributed HashMap implementation in MASS Graph DB, which introduces significant coordination and synchronization overhead that limits the expected benefit of parallelization. Although the single-relation citation graph performs slightly better, the throughput

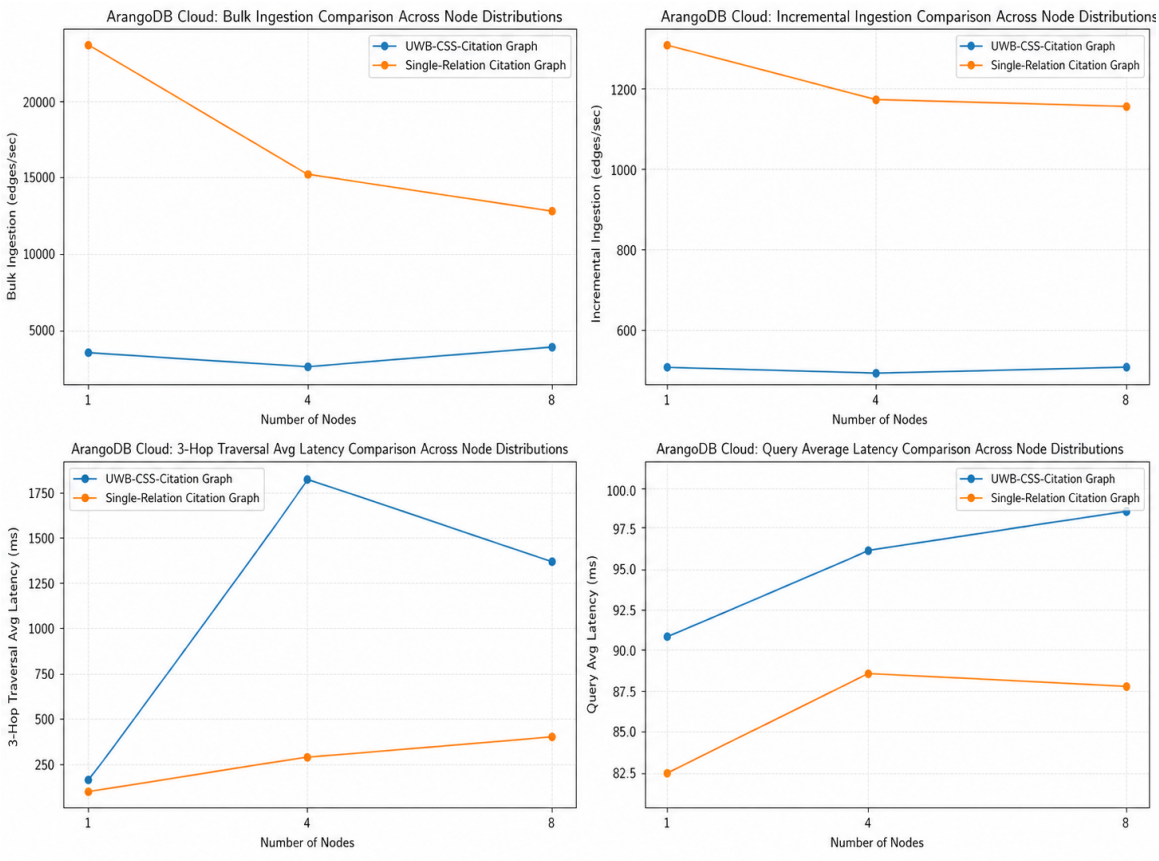


Figure 6.6: Multi-Node ArangoDB Cloud Performance Comparison Between UWB-CSS-Citation and Single-Relation Citation Graphs

values are very close across the tested node counts, indicating that the UWB-CSS graph does not introduce abnormal bulk-loading overhead. This suggests that the additional cost introduced by heterogeneous node and relationship types may be partially offset by the UWB-CSS graph's sparser topology. Therefore, from a quantitative bulk-ingestion perspective, the UWB-CSS Citation Graph aligns well with a larger and denser citation-only graph on MASS Graph DB, supporting its validity as a benchmarkable scholarly graph workload with richer author–work–affiliation–citation semantics.

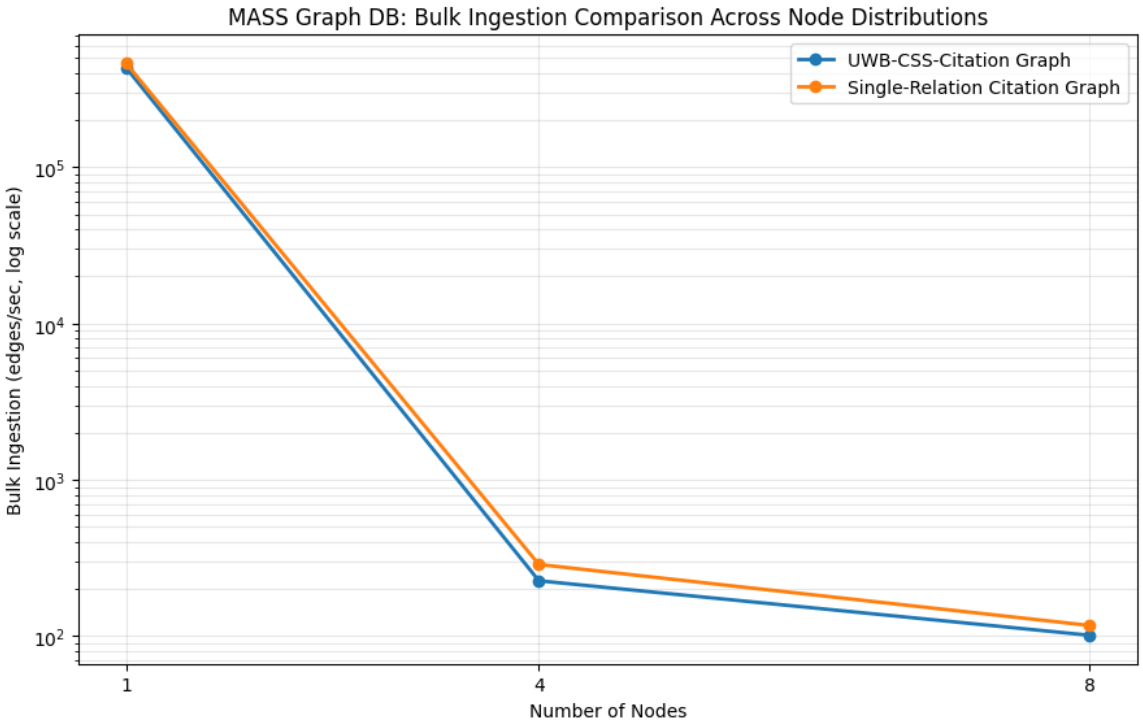


Figure 6.7: Multi-Node MASS Graph DB Performance Comparison Between UWB-CSS-Citation and Single-Relation Citation Graph

6.5 Platform-Specific Performance on the UWB CSS Citation Graph

Table 6.4 shows the performance tests across the three platforms and on a single server. The single-server benchmark results show that each platform benefits different aspects of the

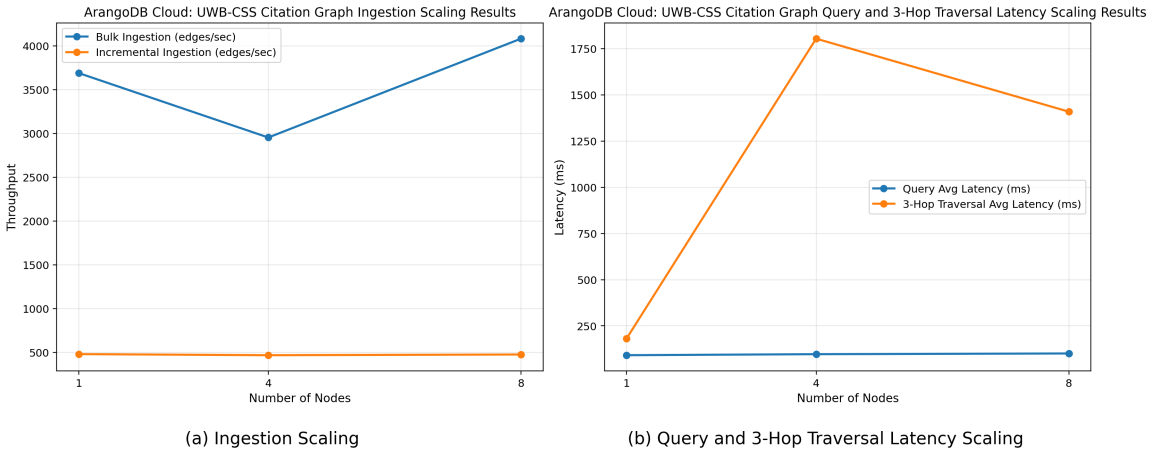
UWB-CSS citation graph workload. MASS Graph DB achieves the strongest ingestion performance, with the highest bulk ingestion and incremental ingestion throughput, indicating that its in-memory execution model is highly effective for write-heavy graph loading tasks. Neo4j provides the best overall query performance, achieving the highest query throughput and the lowest 3-hop traversal latency, making it the strongest fit for interactive scholarly exploration, collaborator discovery, and citation-path analysis. ArangoDB performs more moderately on a single server, with lower ingestion throughput than Neo4j and MASS and higher query latency than Neo4j, but it remains useful as a flexible graph/document backend for cloud-oriented deployment and later distributed evaluation. Overall, these results suggest that MASS benefits ingestion-heavy workloads, Neo4j benefits latency-sensitive and user-facing graph queries, and ArangoDB provides a practical baseline for managed graph storage and scalability experiments.

Table 6.4: Single-node citation graph performance comparison across Neo4j, ArangoDB, and MASS Graph DB for the UWB CSS Citation Graph

Metric	**Neo4j**	**ArangoDB**	**MASS**
Bulk Ingestion Throughput (edges/sec)	9,396.74	4,397.24	432,734.46
Incremental Ingestion Throughput (edges/sec)	843.62	469.37	199,221.13
Query Throughput (QPS)	497.09	108.12	0.1693
Query Avg Latency (ms)	20.08	91.34	5,905.2378
3-Hop Traversal Avg Latency (ms)	7.38	181.98	47,755.2908

The multi-node ArangoDB Cloud results in Figure 6.8 show that the UWB-CSS citation graph responds differently across ingestion, lightweight query, and traversal workloads. For ingestion, bulk loading decreases from 1 node to 4 nodes but then improves at 8 nodes, exceeding the single-node result. This suggests that the distributed configuration begins to provide useful parallelism for bulk loading once the cluster is large enough to offset coordination overhead. Incremental ingestion remains nearly flat across node counts, indicating that smaller update operations gain limited benefit from additional nodes. For read work-

loads, lightweight query latency remains relatively stable, with only a small increase as node count grows, because the query mainly performs node aggregation rather than complex traversal. In contrast, 3-hop traversal latency increases sharply from 1 node to 4 nodes and then decreases at 8 nodes, suggesting that heterogeneous multi-hop traversal is sensitive to distributed execution, but can partially recover when more nodes are available. Overall, ArangoDB Cloud provides the clearest scaling benefit for bulk ingestion, while 3-hop traversal remains more difficult to improve because each traversal may need to access graph data distributed across multiple nodes, adding communication overhead during query execution.



(a) Ingestion Scaling

(b) Query and 3-Hop Traversal Latency Scaling

Figure 6.8: Multi-Node Performance Evaluation of the UWB-CSS Citation Graph on ArangoDB Cloud

The multi-node MASS Graph DB results in Figure 6.9 show that the UWB-CSS Citation Graph benefits most strongly from the single-node in-memory execution configuration rather than distributed deployment. Both bulk ingestion and incremental ingestion achieve very high throughput on one node, but performance drops sharply at 4 nodes and continues to decline at 8 nodes. This suggests that, under the current benchmark configuration, MASS Graph DB's in-memory design is highly effective for local ingestion workloads, while multi-node execution introduces substantial communication, synchronization, or partition-management overhead. Overall, these results indicate that MASS benefits high-throughput

single-node ingestion, but the current distributed setup does not provide scaling benefits for the UWB-CSS citation graph ingestion workload.

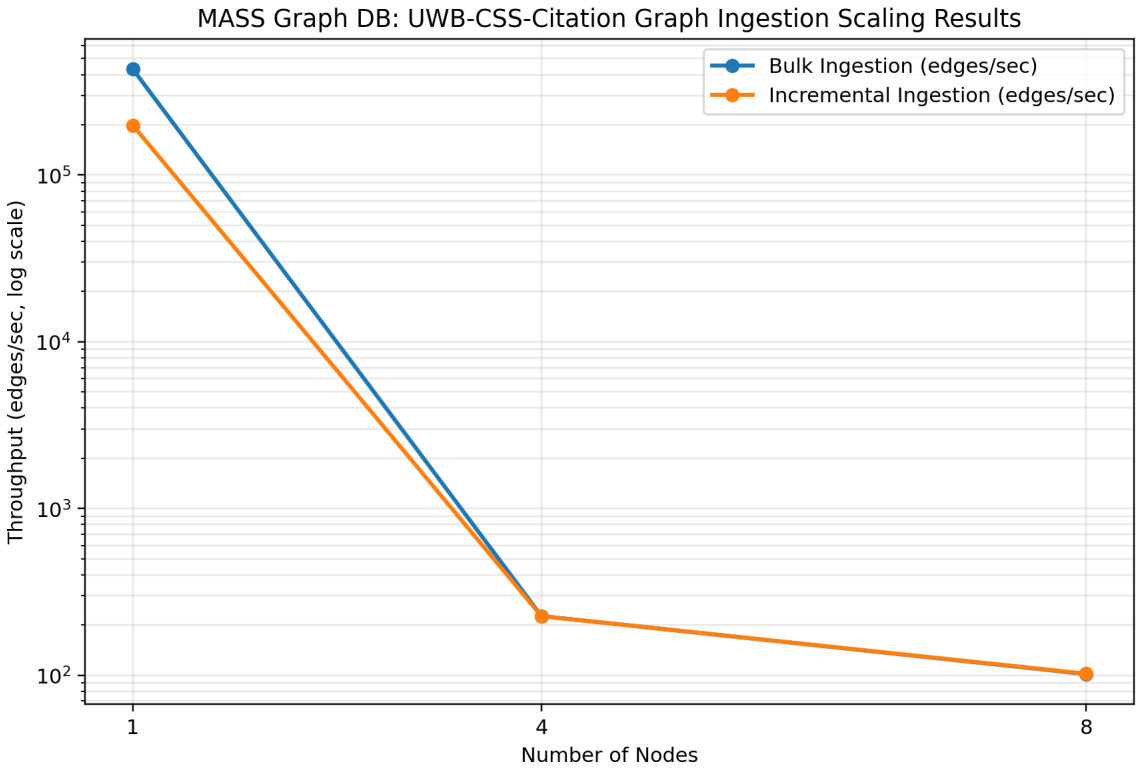


Figure 6.9: MASS Graph DB Multi-Node Ingestion Scaling on the UWB-CSS Citation Graph

Chapter 7

CONCLUSION

This is the final synthesis of the research, summarizing the main contributions, acknowledging the limitations of the current study, and outlining future directions for improving the UWB-CSS citation graph and its benchmarking framework.

7.1 Summary of Contributions

This research hypothesized that a heterogeneous scholarly citation graph integrating authorship, citation, and affiliation relationships can support richer institution-centered scholarly analysis than a single-relation citation graph while still maintaining practical performance under controlled benchmarking conditions. The results support this hypothesis because the proposed graph enables analytical tasks that citation-only graphs cannot support, including collaborator discovery, citation-based research visibility analysis, affiliation-aware filtering, and preliminary identification of external promotion-referee candidates.

The first contribution is the design and construction of a heterogeneous scholarly citation graph that integrates authors, works, affiliations, authorship links, affiliation links, and citation relationships into one unified model. Although UWB CSS faculty are used as the concrete case study, the graph design can be generalized as an institution-centered model for scholarly relationship exploration.

The second contribution is the cross-dataset evaluation of the proposed graph against datasets with different topology, scale, density, and relationship structure. This comparison shows that the UWB-CSS Citation Graph has a richer heterogeneous topology than a single-relation citation graph, but its sparse density helps compensate for the additional structural complexity. In other words, although the graph contains more entity and relationship types, the relatively low edge density limits excessive traversal expansion, helping the graph remain practical for ingestion and interactive multi-hop queries. The multi-node ArangoDB

Cloud results further show that the heterogeneous UWB-CSS Citation Graph can benefit more from distributed scaling than the single-relation citation graph in some workloads, especially when moving from 4 nodes to 8 nodes for bulk ingestion and 3-hop traversal. This suggests that the graph's heterogeneous structure can make better use of additional distributed resources once the cluster is large enough to offset coordination overhead.

The third contribution is the cross-platform benchmarking of the same logical graph across Neo4j, ArangoDB, and MASS Graph DB. The results show that platform suitability is workload-dependent: Neo4j provides the strongest support for interactive query and traversal workloads, MASS Graph DB benefits bulk ingestion and in-memory graph loading, and ArangoDB provides useful cloud-based deployment and scalability support.

Overall, this research shows that the heterogeneous graph model introduces additional construction and traversal cost, but that cost is partly offset by the graph's sparse structure and is justified by the added analytical expressiveness. The multi-node ArangoDB Cloud results further suggest that the heterogeneous UWB-CSS citation graph can benefit from distributed scaling more clearly than the single-relation citation graph for some workloads, especially bulk ingestion and 3-hop traversal when moving from 4 nodes to 8 nodes. Therefore, the proposed graph remains practical on suitable platforms while supporting richer scholarly analysis than structurally simpler citation-only graphs.

7.2 Limitations

The first limitation is data completeness. The graph depends primarily on OpenAlex metadata, which may contain missing publications, incomplete affiliations, or ambiguous author identities, even though manual validation and correction are used to reduce these errors.

The second limitation is scope. The current graph is seeded from UWB CSS faculty, so the results may not fully generalize to larger scholarly communities, other departments, or cross-institution graphs without further expansion and testing.

The third limitation is benchmark comparability. Neo4j, ArangoDB, and MASS Graph DB use different storage models, execution models, and persistence mechanisms, so some metrics must be interpreted as comparable workload measurements rather than perfectly identical internal operations.

7.3 Future Work

Future work should expand the graph beyond UWB CSS by including more faculty from other UWB departments, additional institutions, and broader cross-institution scholarly communities. This expansion would improve the representativeness of the graph and allow the system to support richer comparisons across departments, institutions, and research areas. In addition, each Work node can be enriched with topic-related metadata from OpenAlex, such as topic, subfield, and domain. These properties would allow users to filter collaborator, citation, and referee-candidate searches by research area, making the graph more useful for field-aware scholarly analysis.

Future work should also improve data quality by adding missing publications, refining author disambiguation, updating affiliation records, and continuing manual validation against external scholarly sources. The graph can also be enriched with institutional classification metadata, such as Carnegie Classification attributes, so that citing institutions can be analyzed by institution type, size, research intensity, and institutional profile. This would allow the system to answer questions such as whether UWB-related papers are cited mainly by peer institutions, larger research universities, teaching-focused institutions, or other organization types.

The benchmarking framework should be extended with more datasets, larger graph scales, and more aligned query workloads so that platform behavior can be evaluated more comprehensively across topology, density, schema heterogeneity, and distributed deployment conditions. Future evaluation can also include an ablation study where heterogeneity is turned "on" and "off" by comparing the full heterogeneous graph with simplified projections, such as author–author co-authorship graphs or paper–paper citation graphs. This would help isolate the performance cost and analytical benefit of heterogeneous modeling. In addition, adjustable or parameterized datasets, such as datasets that vary graph size, density, number of node types, number of edge types, citation depth, co-authorship frequency, and query depth, could be used to test multiple parameter combinations more systematically.

Finally, future work could extend the graph into an interactive scholarly analytics tool

that allows faculty and administrators to explore collaborators, citation influence, research communities, and potential external referee candidates through a user-facing interface.

BIBLIOGRAPHY

[1] Distributed Systems Laboratory, University of Washington Bothell, "Mass: A parallelizing library for multi-agent spatial simulation," 2026. Accessed: Mar. 3, 2026. [Online]. Available: https://depts.washington.edu/dslab/MASS/.

[2] Y. Hong and M. Fukuda, "Pipelining graph construction and agent-based computation over distributed memory," 2022. Accessed: Mar. 3, 2026. [Online]. Available: https://faculty.washington.edu/mfukuda/papers/biggraphs22.pdf.

[3] M. Ma, M. Fukuda, and J. Shi, "Toward implementing an agent-based distributed graph database system," *KN Big Data*, 2024.

[4] H. Rajvaidya. "An agent-based graph database," Accessed: Mar. 3, 2026. [Online]. Available: https://depts.washington.edu/dslab/MASS/reports/HarshitRajvaidya_whitepaper.pdf.

[5] M. Dea. "An agent-based graph database benchmarking program," Accessed: Mar. 3, 2026. [Online]. Available: https://depts.washington.edu/dslab/MASS/reports/MichelleDea_whitepaper.pdf.

[6] A. R. Prajapati. "An enhancement of distributed graph queries in an agent-based graph database," Accessed: Mar. 3, 2026. [Online]. Available: https://depts.washington.edu/dslab/MASS/reports/AatmanPrajapati_thesis.pdf.

[7] Neo4j, Inc. "What is neo4j?" Neo4j Documentation, Accessed: Mar. 3, 2026. [Online]. Available: https://neo4j.com/docs/.

[8] Neo4j. "Scale a neo4j deployment," Accessed: May 12, 2026. [Online]. Available: https://neo4j.com/docs/operations-manual/current/kubernetes/operations/scaling/.

[9] Neo4j. "The neo4j graph platform: Index-free adjacency," Accessed: May 12, 2026. [Online]. Available: https://neo4j.com/graphacademy/training-overview-40/02-overview40-neo4j-graph-platform/.

[10] ArangoDB. "Graphs," Accessed: May 12, 2026. [Online]. Available: https://docs.arangodb.com/3.12/graphs/.

[11] ArangoDB. "Cluster," Accessed: May 12, 2026. [Online]. Available: https://docs.arangodb.com/3.12/components/web-interface/cluster/.

[12] ArangoDB. "Smartgraphs," Accessed: May 12, 2026. [Online]. Available: https://docs.arangodb.com/3.12/graphs/smartgraphs/.

[13] ResearchGate. "Researchgate," ResearchGate GmbH, Accessed: Mar. 3, 2026. [Online]. Available: https://www.researchgate.net/.

[14] Academia.edu. "Academia.edu," Academia, Inc., Accessed: Mar. 3, 2026. [Online]. Available: https://www.academia.edu/.

[15] V. Yadav et al., "Exploring citation and author collaboration network with graph database," *ResearchGate*, 2024.

[16] N. Roozbahani et al., "Presenting a dataset for collaborator recommending systems in academic social networks: A case study on researchgate," *arXiv preprint arXiv:2101.01141*, 2021.

[17] B. Adam et al., "What comes first: The co-authorship network or the citation?" *ResearchGate*, 2017.

[18] S. Ghosh, "Examining different research communities: Authorship network," *arXiv preprint arXiv:2409.00081*, 2025.

[19] Connected Papers. "Connected papers: Visual graph explorer for academic papers," Accessed: Mar. 15, 2026. [Online]. Available: https://www.connectedpapers.com/.

[20] P. K. Behera, S. J. Jain, and A. Kumar, "Visual exploration of literature using connected papers: A practical approach," *Issues in Science and Technology Librarianship*, 2023.

[21] C. Betts, R. Power, and W. Ammar, "Grapal: Connecting the dots in scientific literature," *arXiv preprint arXiv:1902.05170*, 2019.

[22] J. Priem, H. Piwowar, and R. Orr, "Openalex: A fully-open index of scholarly works, authors, venues, institutions, and concepts," *arXiv preprint arXiv:2205.01833*, 2022. DOI: 10.48550/arXiv.2205.01833.

[23] Graph Data Council. "Ldbc social network benchmark (ldbc snb)," Graph Data Council, Accessed: Mar. 23, 2026. [Online]. Available: https://ldbcouncil.org/benchmarks/snb/.

[24] Stanford Network Analysis Project, *Stanford Large Network Dataset Collection*, https://snap.stanford.edu/data/, Accessed: May 9, 2026.