Contents lists available at ScienceDirect

SoftwareX

journal homepage: www.elsevier.com/locate/softx

Original software publication

APOTHEOSIS: An efficient approximate similarity search system

Daniel Huici, Ricardo J. Rodríguez [*], Eduardo Mena

Dpto. de Informática e Ingeniería de Sistemas, Universidad de Zaragoza, Spain

ARTICLE INFO

Keywords:
Approximate search methods
Approximate K-nearest neighbors
Approximate matching
Similarity digest algorithms
Data similarity analysis

ABSTRACT

APOTHEOSIS is a tool for efficiently identifying and comparing data similarity in large datasets, addressing challenges faced by traditional methods such as scalability and speed. APOTHEOSIS overcomes them by combining advanced algorithms and data structures, enabling fast and accurate similarity analysis. Specifically, it uses a custom hierarchical small navigation world as an approximate K-nearest neighbors search method, and approximate similarity digests algorithms to find common features between similar data items, also supporting various distance metrics beyond vector-based approaches. Our software tool is designed for seamless integration into research workflows, improving reproducibility and facilitating the comparison of large-scale, high-dimensional data comparison across multiple domains.

Code metadata

Current code version	v1.2
Permanent link to code/repository used for this code version	https://github.com/ElsevierSoftwareX/SOFTX-D-24-00557
Permanent link to Reproducible Capsule	–
Legal Code License	GPLv3
Code versioning system used	git
Software code languages, tools, and services used	Python
Compilation requirements, operating environments & dependencies	Python (≥ 3.9.2), Flask, matplotlib, mysql-connector-python, networkx, numpy, pandas, py_tlsh, pytest_regressions, PyYAML, Requests, scipy, seaborn, sensitivity, SQLAlchemy, statsmodels, waitress, Werkzeug
If available Link to developer documentation/manual	https://github.com/reverseame/APOTHEOSIS/blob/main/README.md
Support email for questions	reverseame@unizar.es

1. Motivation and significance

APOTHEOSIS (*APprOximaTe searcH systEm Of Similarity dIgeSts*) is a tool designed to efficiently identify and compare data similarity in large datasets, an essential capability in fields such as digital forensics, data deduplication, or genomics, to name a few. For instance, a rapid identification of similar binary artifacts can significantly expedite incident response investigations [1].

Traditional methods of data comparison often struggle with scalability and speed when working with large amounts of data (known as *the curse of dimensionality* [2]). Our tool addresses this challenge by leveraging advanced algorithms and efficient data structures to reduce analysis time and computational demands. Specifically, it combines *approximate search techniques* [3] to quickly find close (but not exact) matches and *approximate similarity matching algorithms* [4] to identify shared features between similar artifacts.

APOTHEOSIS improves scientific research by offering a powerful platform for data similarity analysis, easily integrated into various workflows. It improves the speed and accuracy of large-scale data comparisons, which is critical for timely decision making and hypothesis testing. The tool also ensures reproducibility in scientific studies by providing a systematic approach to data analysis.

APOTHEOSIS is designed with a modular architecture, making it accessible to users from various fields. After downloading [5] and installing the tool, users can analyze datasets such as system logs, genomic sequences, or document collections. The data may require preprocessing to ensure it is in a suitable format, including cleaning

[*] Corresponding author.
E-mail address: rjrodriguez@unizar.es (R.J. Rodríguez).

https://doi.org/10.1016/j.softx.2024.102016
Received 18 October 2024; Received in revised form 2 December 2024; Accepted 11 December 2024
Available online 24 December 2024

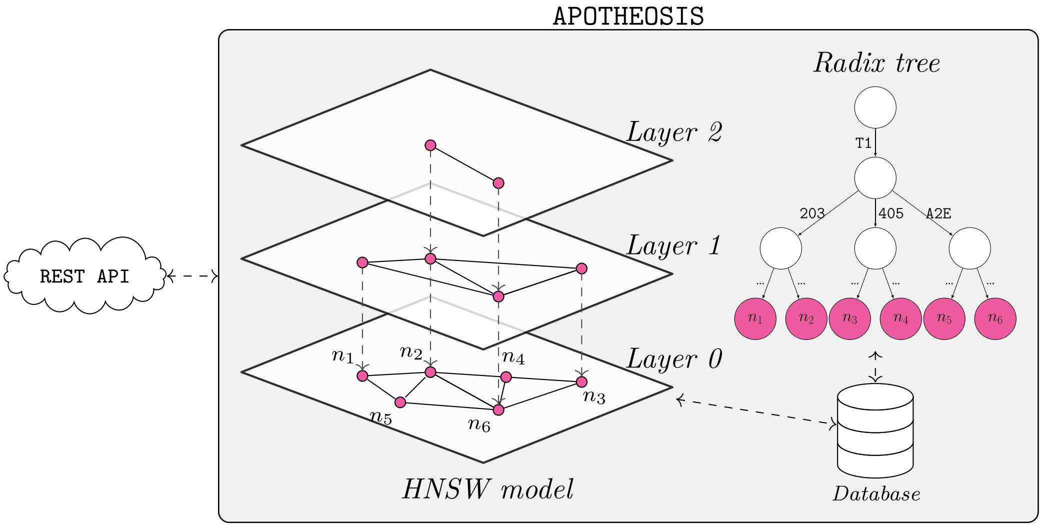


Fig. 1. High-level overview of APOTHEOSIS.

and feature extraction. The similarity digests [4] of the data are then used in the APOTHEOSIS model to perform similarity searches.

Related work

K-nearest neighbor search (K-NNS) is a common method to find the nearest items in a dataset based on a defined distance function. However, K-NNS faces challenges with high-dimensional data, where exact solutions may be computationally intensive [6]. To address this problem, K-nearest approximate neighbor search (K-ANNS) was introduced [3], which finds data points that are close to the query, but not necessarily the closest, making it more practical and efficient for high-dimensional spaces.

APOTHEOSIS employs a graph-based K-ANNS method that uses a custom implementation of Hierarchical Small Navigation World (HNSW) [7] for approximate search. HNSW is popular for its efficiency in approximate nearest neighbor searches. Many HNSW implementations focus particularly in vector searches. The most widely used implementation is HNSWlib [8], a lightweight C++ library with Python bindings, which offers customizable distance functions, albeit limited to Euclidean distance, inner product, and cosine similarity in Python.

Another notable tool is Facebook AI Similarity Search [9], a C++ implementation with GPU acceleration support that uses vector-based search functions. Various HNSW implementations exist for languages like Java, Go, and Rust, but they also rely on vector search. A Python3 implementation of HNSW was recently released [10], but it shares the same limitations as the other solutions.

Unlike vector-based approaches, which may struggle with large datasets, APOTHEOSIS is built to handle large volumes of data using any type of distance metric or similarity score.

2. Software description

APOTHEOSIS is an approximate search system that uses similarity digest algorithms (SDA) [4] to compare digital artifacts based on similarity rather than exact matches. It employs two key data structures: a space-saving trie for efficient item searching (specifically, a radix tree [11]) and a custom implementation of HNSW [7], a graph-based method for fast, approximate nearest neighbor searching in high-dimensional spaces.

Our system is highly extensible and adaptable to various fields, including digital forensics, genomics, and environmental science, to name a few. It significantly improves the ability to perform data similarity analysis, which is crucial for various scientific and analytical tasks. For instance, in digital forensics, it enables analysts to quickly identify and correlate evidence across multiple devices, speeding up investigations and uncovering coordinated malicious activity. Likewise, in genomics, it helps researchers study evolutionary relationships by comparing genetic data across species, improving the efficiency of scientific analysis.

2.1. Software architecture

Fig. 1 provides a high-level overview of APOTHEOSIS. Our system leverages two complementary data structures – a radix tree and a custom HNSW – alongside similarity digests to efficiently manage and compare data. The APOTHEOSIS database stores all hashes and associated metadata, such as log or document filenames, which helps maintain the spatial efficiency of the HNSW structure. Additionally, APOTHEOSIS provides a REST API, enabling seamless integration with other tools and systems for automated and real-time data analysis.

Both data structures in APOTHEOSIS were implemented as generic Abstract Data Types (ADTs) [12]. Following software engineering best practices [13], we designed these structures to be easily extensible for any similarity function or data type. Specifically, we used the *Abstract Factory* design pattern to define interfaces for creating related objects and the *Strategy* pattern to allow dynamic selection of behaviors at runtime. In APOTHEOSIS, the Strategy pattern manages the specific details of a node, while the Abstract Factory pattern handles different distance and similarity metrics.

To support open science and foster research, we have released APOTHEOSIS under the GNU/GPLv3 license. Our implementation includes typical ADT operations such as *initialization*, *insertion*, *deletion*, *lookup*, and *search*. We adhere to Python best practices and PEP8 style guidelines [14] for ease of maintenance and readability. The source code is managed using Git and is available in our GitHub [5].

2.2. Software functionalities

In this section, we describe the key operations of our system: initialization, insertion, deletion, lookup, and search. Each operation is designed to maintain system consistency, optimize performance, and support flexible similarity analysis, as detailed below.

Algorithm 1: Insertion of a new node into an Apotheosis model.

```
Input: Apotheosis model $\mathcal{M} = \{$Radix Tree $\mathcal{RT}$, HNSW $\mathcal{H}\}$, new node $n$ with hash $h_n$
1 Procedure InsertNode($\mathcal{M}$, $n$):
2     $existing\_node \leftarrow$ SearchRadixTree($\mathcal{RT}, h_n$)
3     if $existing\_node \neq null$ then
          // Node already exists, avoid redundant insertion
4         return
5     InsertRadixTree ($\mathcal{RT}, h_n, n$);
6     InsertHNSW ($\mathcal{H}, n$) // Follows the algorithm in [7]
```

2.2.1. Initialization operation

The radix tree and HNSW data structures are initialized to handle a specific SDA to ensure consistent results by preventing the mixing nodes from different SDA. The radix tree, which uses hashes as keys and pointers to corresponding nodes in the HNSW structure as values, starts as an empty tree. HNSW requires different configuration parameters [7]:

- Number of established connections (M): M determines the number of connections a new node will establish upon insertion.
- Maximum number of neighbors (M_{max}): M_{max} sets the upper limit on the number of neighbors a node can have at each layer, except for the ground layer which uses a separate parameter, M_{max0}.
- Neighbors to explore (ef), which is how many neighbors are explored during both the construction and search phases.

The values of these parameters affect the behavior and performance of HNSW. For instance, increasing M_{max} may improve search recall, but can also increase memory usage and query time. Similarly, a higher ef value can result in a more accurate index, but may increase construction time. Users should evaluate these parameters based on their specific problem requirements. To assist with this, we provide scripts for conducting sensitivity analysis on these parameters under various configurations.

2.2.2. Insertion operation

Once both the radix tree and HNSW structures have been initialized, adding new nodes to the system follows a systematic approach (see Algorithm 1). The `InsertNode` procedure first checks whether the new hash already exists by searching for it in the radix tree (line 2). If found, the node is not added again, avoiding redundancy (lines 3–4). Otherwise, the new node is inserted first into the radix tree (line 5) and then into the HNSW structure (line 6). The insertion operation in HNSW follows the algorithm given in [7]. This dual data structure approach mitigates the risk of false negatives that can occur with HNSW alone, by ensuring that the same node is not inserted multiple times.

Let us finally remark that although our system specifically uses similarity digests, the HNSW structure is designed to handle any type of element as long as a comparison method is available.

2.2.3. Delete operation

The delete operation in APOTHEOSIS involves deleting a node from the radix tree and the HNSW graph structure (see Algorithm 2). This process ensures that the node is completely detached from both data structures while maintaining the integrity and functionality of the overall structure.

It first verifies that the node can be safely removed with some sanity checks (e.g., checking conditions such as that the structure is not empty and that the node's distance algorithm matches the structure's algorithm; lines 2–3). If these conditions are met, the node is removed from the radix tree (line 4). If the node is found, it is then removed from the HNSW graph structure (line 7). To do this, it iterates through all layers of the node, removing the node from the lists of its neighbors (and vice versa), and then removes it from the HNSW structure's internal dictionary. If the node is the only one in its layer, the layer key is also removed from the internal dictionary. The final step involves, if necessary, updating the entry point of the HNSW graph: if the node to be deleted is the current entry point, it searches for a new entry point among the node's neighbors at the highest layer of the current entry point. If no neighbors are found, it moves to the next immediate lower layer, searching for the first neighbor in a lower layer of the node that was the entry point to set it as the new entry point (lines 8–11).

2.2.4. Lookup and search operations

Given a query key (digest), our system first checks the radix tree, after some sanity checks (specifically, the distance algorithm of the query and the system must match). If the key is found, it retrieves the corresponding node and its KNN from the HNSW structure. Otherwise, HNSW is used to perform an approximate search to find the nearest nodes. This approach ensures that even if the exact key is missing, we can still obtain the approximate nearest neighbors. Specifically, APOTHEOSIS supports two main search operations, explained below.

Nearest Neighbors-Based Search. The KNN search operation (see Algorithm 3) begins by initializing a priority queue with the entry point of the highest layer (line 2). Given a query node q and parameter $K \in \mathbb{N}_{>0}$, the search process begins at the top layer and traverses down to the ground layer (lines 3–9). At each layer, the neighbors of the current entry point are evaluated based on their distance to the query node, and the priority queue is updated with any newly found closer nodes (lines 5–7). The entry point for the next lower layer is selected from the nearest neighbors in the queue (line 9). Finally, the K nearest neighbors are returned from the priority queue (line 10). Unlike [7], our method extends this set by also considering the neighbors of the found nodes, selecting those closest to the query node.

Threshold-Based Search. The threshold-based search operation (see Algorithm 4) takes as input a query node q, a threshold value $t > 0$, and a comparison mode m (distance metric or similarity score). It identifies nodes whose similarity score to the query node is strictly greater or lower than the threshold, depending on the mode. The search proceeds similarly to the KNN search, starting from the topmost layer and moving down to the ground layer, but with one key difference: nodes are added to the candidate set only if they meet the threshold criteria (lines 7–8). After reaching the ground layer, the final candidate set is expanded by considering neighbors that also meet the threshold condition (line 11).

2.2.5. REST API interface

We have developed a REST API interface for APOTHEOSIS to simplify similarity analysis and extend its accessibility. This interface allows for easier integration of APOTHEOSIS into existing workflows and solutions.

The REST API presents two endpoints for search operations on a loaded HNSW model, allowing users to choose between `knn` or `threshold` searches. Users provide the required search data and the API returns a JSON response indicating whether an exact match was found and listing approximate neighbors for the query hash.

Algorithm 2: Deletion of a node from an Apotheosis model.

```
Input: Apotheosis model $\mathcal{M}$ = {Radix Tree $\mathcal{RT}$, HNSW $\mathcal{H}$}, node $n$ with hash $h_n$
Procedure DeleteNode($\mathcal{M}$, $n$):
    // Sanity checks before deletion
    if structure is empty or distance algorithm mismatch then
        return
    $removed \leftarrow$ RemoveRadixTree($\mathcal{RT}, h_n$)
    if $removed = false$ then
        // Node not found in Radix Tree
        return
    RemoveHNSW ($\mathcal{H}, n$)
    // Update entry point if necessary
    if $n$ is the current entry point in $\mathcal{H}$ then
        $new\_entry \leftarrow$ FindNewEntryPoint(GetNeighbors($n$, highest layer))
        if $new\_entry = null$ then
            $new\_entry \leftarrow$ Find first neighbor in lower layers
```

Algorithm 3: KNN search of a node in an Apotheosis model.

```
Input: Apotheosis model $\mathcal{M}$ = {Radix Tree $\mathcal{RT}$, HNSW $\mathcal{H}$}, query node $q$ with hash $h_q$, parameter $K \in \mathbb{N}_{>0}$
Procedure KNNSearch($\mathcal{M}$, $q$, $K$):
    $queue \leftarrow$ PriorityQueue with entry point of highest layer
    $current\_layer \leftarrow$ highest layer
    while $current\_layer \geq$ ground layer do
        $neighbors \leftarrow$ GetNeighbors(entry point, $current\_layer$)
        foreach neighbor $n$ in $neighbors$ do
            UpdateQueue ($queue, n, q$)
        $current\_layer \leftarrow current\_layer - 1$
        $entry\_point \leftarrow$ Get nearest neighbor from queue
    return Top $K$ nodes from the priority queue
```

Algorithm 4: Threshold search of a node in an Apotheosis model.

```
Input: Apotheosis model $\mathcal{M}$ = {Radix Tree $\mathcal{RT}$, HNSW $\mathcal{H}$}, query node $q$ with hash $h_q$, threshold $t > 0$, comparison mode $m$
Procedure ThresholdSearch($\mathcal{M}$, $q$, $t$, $m$):
    $queue \leftarrow$ PriorityQueue with entry point of highest layer
    $current\_layer \leftarrow$ highest layer
    while $current\_layer \geq$ ground layer do
        $neighbors \leftarrow$ GetNeighbors(entry point, $current\_layer$)
        foreach neighbor $n$ in $neighbors$ do
            if ThresholdCheck ($n$, $q$, $t$, $m$) then
                UpdateQueue ($queue, n, q$)
        $current\_layer \leftarrow current\_layer - 1$
        $entry\_point \leftarrow$ Get nearest neighbor from queue
    $final\_candidates \leftarrow$ Get nodes from the priority queue that meet threshold return $final\_candidates$
```

3. Illustrative example

To demonstrate the capabilities and potential applications of APOTHEOSIS, we perform a text similarity analysis in subsets of chapters from Miguel de Cervantes' "Don Quixote", focusing on comparing the performance of TLSH and ssdeep. We incrementally extracted text from the first 10 chapters of the novel by creating chunks that included progressively more chapters. Specifically, we first extracted chapter 1, then created a chunk containing chapters 1 and 2, followed by a chunk containing chapters 1, 2, and 3, and continued this process until all 10 chapters were included. We then used APOTHEOSIS to assess the similarity between these progressively larger chunks. Table 1 shows the hash values for each chapter. To quantify similarity, Table 2 presents the percentage representation of the byte size of each smaller file relative to progressively larger files. The values indicate what proportion of the byte size of the larger file is accounted for by the smaller file, providing an indication of their similarity in terms of content overlap.

We implemented a class, ChapBookHashNode, within APOTHEOSIS to handle and analyze these hashes, and used the draw method to generate visualizations of the HNSW model layers. This class should be a specialized subclass of HashNode, a component of the APOTHEOSIS framework. Its full code and the driver script can be found in the chapbooks-example branch.[1]

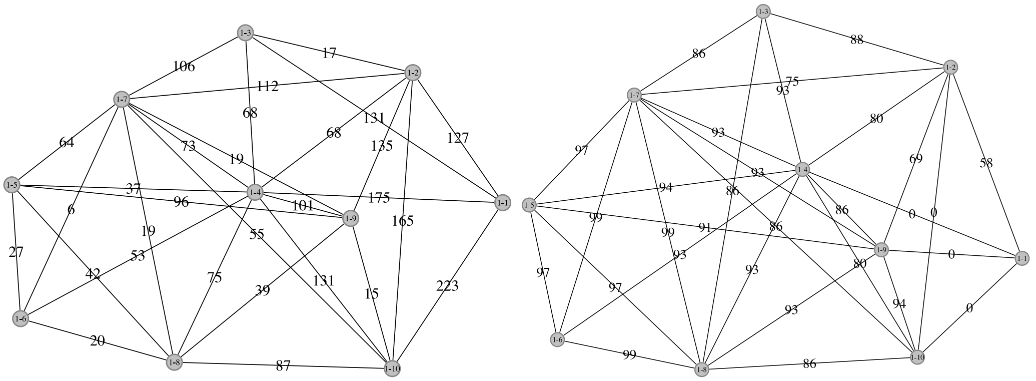
[1] See https://github.com/reverseame/APOTHEOSIS/tree/chapbooks-example.

Table 1
TLSH and ssdeep hashes from chapter extracts from the book “Don Quixote”.

Chapters	TLSH (first line) and ssdeep (second line) hashes
1–1	T109217706ADC031F708D313D2C756D997D54192847604C2DAD9B6476630C57CCC9AFD48 24:2VEzWputOLBG6Keyvcj2FRd/LCfg3oZwamkMjyLyFJWV/K6q/xNn:2AN+GjdzHdzSAoZwYMHF0VCxN
1–2	T14F5194239DC013AB48D31386D686D6B3E080E6C07658C2EBD976D24A31C96CCCBAFD48 48:2AN+GjdzHdzSAoZwYMHF0VCx4byL1vTOZCgqF+SjtHBwLhmxSXMmz2bLAYK:2ANb59jl08xtL1vTLgqbthwLhmxSXMaz
1–3	T1E35194279DC013AB44D31386D64AC6B3E484A6C07758C2E7D976D64A30CD68CDBAFE48 96:2ANb59jl08xtL1vTLgqbthwLhmxSXMa2bLl3XSGc:V99j6OwqbthUmxSdeLVA
1–4	T117818323DDC413AB85D31386E78BD2A3F484A5C07654C1EB9566864A30CC6CCCBEFD88 96:2ANb59jl08xtL1vTLgqbthwLhmxSXMa2bLl3XSGYJe0XQD:V99j6OwqbthUmxSdeLVOenD
1–5	T1659163239EC017AB81E32385D78BD663F584A5C07254C1D7A566C74A30CC78CDBAFE88 96:2ANb59jl08xtL1vTLgqbthwLhmxSXMa2bLl3XSGYJe0XQFaVBx:V99j6OwqbthUmxSdeLVOenkVBx
1–6	T14CA184279DC017AB81E32385D38BD663F584A5D07255C1E7E966C24A31CC68CDBAFE88 96:2ANb59jl08xtL1vTLgqbthwLhmxSXMa2bLl3XSGYJe0XQFaVBbIy:V99j6OwqbthUmxSdeLVOenkVBbIy
1–7	T1FEB18527DDC017AB81E32385E38BD663F584A5D07255C1E7D966C24A31CC68CDBAFE88 96:2ANb59jl08xtL1vTLgqbthwLhmxSXMa2bLl3XSGYJe0XQFaVBbIZ:V99j6OwqbthUmxSdeLVOenkVBbIZ
1–8	T139B164279DC0176B81E32385E38BD663F584A5D07255C1E7D566C24E31CC68CDBAFD88 96:2ANb59jl08xtL1vTLgqbthwLhmxSXMa2bLl3XSGYJe0XQFaVBbIX:V99j6OwqbthUmxSdeLVOenkVBbIX
1–9	T1EEC186279DC41B6B81E32385E78BDA73F484A5C07215C1DBD966C24E21C868CDBAFD88 96:2ANb59jl08xtL1vTLgqbthwLhmxSXMa2bLl3XSGYJe0XQFaVBbIoJv7jmmW:V99j6OwqbthUmxSdeLVOenkVBbIADjpW
1–10	T1ECD197279DC41B6B81E32385D38BC973F584A5D07255C1DBA96AC24E21CC68CDBAFD88 192:V99j6OwqbthUmxSdeLVOenkVBbIADjpN69v:zJ6OwqbthUmIUJZnibIIN69v

Table 2
Percentage representation of smaller files within larger files, based on byte size.

–	1–1	1–2	1–3	1–4	1–5	1–6	1–7	1–8	1–9	1–10
1–1	100.00	41.44	38.82	31.41	26.74	24.38	24.09	23.45	21.56	19.70
1–2	–	100.00	93.68	75.78	64.53	58.84	58.12	56.59	52.02	47.53
1–3	–	–	100.00	80.90	68.88	62.81	62.04	60.41	55.53	50.74
1–4	–	–	–	100.00	85.15	77.64	76.70	74.68	68.64	62.72
1–5	–	–	–	–	100.00	91.18	90.07	87.70	80.62	73.66
1–6	–	–	–	–	–	100.00	98.78	96.18	88.41	80.78
1–7	–	–	–	–	–	–	100.00	97.36	89.50	81.78
1–8	–	–	–	–	–	–	–	100.00	91.92	83.99
1–9	–	–	–	–	–	–	–	–	100.00	91.37
1–10	–	–	–	–	–	–	–	–	–	100.00



(a) HNSW’s layer 0 with TLSH (b) HNSW’s layer 0 with ssdeep

Fig. 2. Ground layers of the HNSW of each APOTHEOSIS model.

Figs. 2(a) and 2(b) illustrate the HNSW of each APOTHEOSIS model constructed for each hashing algorithm in this running example. Our system allows for the extraction of DOT files for each layer, containing detailed information about nodes, edges, and any additional custom data defined by the user. This feature provides a comprehensive representation of the structure and relationships within each layer, allowing for better analysis based on specific requirements. Note also that the ground layers differ due to the probabilistic nature of the underlying HNSW model. When nodes are inserted, they may reach layer 0 through different entry points, leading to variations in their nearest neighbors. This variability directly influences the structure and neighborhood relations in the resulting graph. As a result, even though nodes remain the same, the connections established between them may vary.

The results indicate differences in performance: ssdeep shows limited similarity between some chapters, while TLSH reflects a wider range of differences. For instance, Chapters 1–10 and 1–9 have low TLSH distance and high ssdeep similarity (i.e., both agree that they have significant similarity). Both algorithms also agree on the high similarity between Chapters 1–7 and 1–8. According to Table 2, the percentage of byte size of Chapter 1–6 within Chapter 1–7 is 98.78%, indicating that Chapter 1–6 accounts for almost all of the content of Chapter 1–7. This similarity is also corroborated by both HNSW models, as they show a lower TLSH score (6) and a higher ssdeep score (99), further supporting the strong overlap between these chapters. This example demonstrates how our tool can effectively compare and analyze textual data across different hashing techniques.

4. Impact

APOTHEOSIS improves research by accelerating, improving the accuracy, and scaling data analysis. It enables faster large-scale similarity searches, providing timely insights and decisions. Its ability to handle large data sets helps researchers identify trends and patterns in big data that were previously difficult to detect.

Our custom implementation of HNSW and radix tree significantly reduce the computational burden of similarity searches. In particular, HNSW achieves a worst-case complexity of $\mathcal{O}(N \log N)$ for constructing the graph and $\mathcal{O}(\log N)$ for querying [7]. Conventional methods, such as pairwise comparisons, that compare every item in the dataset against all others, while straightforward, exhibits a worst-case computational complexity of $\mathcal{O}(N^2)$. This improvement is achieved thanks to the hierarchical layers and optimized connectivity management, as the number of comparisons required are reduced. Similarly, the radix tree structure further accelerates operations such as insertion, deletion, and search by organizing data based on common prefixes. Operations on the radix tree exhibit a complexity of $\mathcal{O}(L)$, where L is the key length [11]. These combined complexities ensure scalability to large data sets.

Our software tool also opens up new research opportunities by enabling efficient and accurate data similarity analysis and by helping to address innovative questions. For instance, *how large-scale similarity analysis of digital artifacts can improve the detection and understanding of complex cyberattacks*? (digital forensics), *what previously unidentified genetic patterns or relationships can be discovered by comparing large genomic datasets*? (genomics), or *how analyzing ecological data from multiple regions and time periods can improve our understanding of the impacts of climate change*? (environmental science).

Finally, APOTHEOSIS's REST API enables integration into automated workflows, reducing manual effort and increasing efficiency. This enables real-time data similarity analysis, which is essential for applications such as cybersecurity and digital forensics, to name a few. Additionally, its modular architecture makes it easy to use across multiple domains, fostering a more integrated approach to data analysis.

5. Conclusions

Efficiently handling and scrutinizing large datasets poses a significant hurdle in various domains, underscoring the critical need for fast and accurate tools to quickly get insights and make decisions. In this paper, we have introduced APOTHEOSIS, an extensible and versatile system that leverages the power of approximate search methods and similarity digest algorithms to facilitate similarity data analysis. This combination allows for fast and efficient identification of similar digital artifacts within large data sets. To improve usability and accessibility, we have also provided a REST API interface for APOTHEOSIS, allowing forensic analysts to seamlessly integrate it into existing research and analytical workflows.

As future work, we intend to explore the integration and comparative analysis of other K-ANN search methods, in addition to HNSW, to evaluate their impact on performance, scalability, and accuracy in various data similarity tasks.

CRediT authorship contribution statement

Daniel Huici: Writing – review & editing, Writing – original draft, Software, Investigation, Conceptualization. **Ricardo J. Rodríguez:** Writing – review & editing, Writing – original draft, Visualization, Supervision, Software, Resources, Project administration, Investigation, Funding acquisition, Conceptualization. **Eduardo Mena:** Writing – review & editing, Visualization, Validation, Methodology.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgments

This research was supported in part by the Spanish project TED2021-131115A-I00, funded by MCIN/AEI/10.13039/501100011033, and by the Recovery, Transformation and Resilience Plan funds, financed by the European Union (Next Generation). The research of R. J. Rodríguez was also supported by the Recovery, Transformation and Resilience Plan funds, financed by the European Union (Next Generation), and by the Spanish National Cybersecurity Institute (INCIBE) under *Proyecto Estratégico CIBERSEGURIDAD EINA UNIZAR*. The research of E. Mena by the Spanish project PID2020-113903RB-I00 (AEI/FEDER, UE). Additionally, the research of R. J. Rodríguez and E. Mena was supported by the University, Industry and Innovation Department of the Aragonese Government under *Programa de Proyectos Estratégicos de Grupos de Investigación* (DisCo and SID research groups, refs. T21-23R and T42-23R, respectively).

References

[1] Johansen G. Digital forensics and incident response: incident response tools and techniques for effective cyber threat response. 3rd ed. Packt Publishing; 2022.

[2] Bellman RE. Dynamic programming. Courier Corporation; 2003.

[3] Indyk P, Motwani R. Approximate nearest neighbors: Towards removing the curse of dimensionality. In: Proceedings of the thirtieth annual ACM symposium on theory of computing. New York, NY, USA: Association for Computing Machinery; 1998, p. 604–13.

[4] Breitinger F, Guttman B, McCarrin M, Roussev V, White D. Approximate matching: definition and terminology. Techreport NIST special publication 800-168, National Institute of Standards and Technology; 2014.

[5] Huici D, Rodríguez RJ. Apotheosis (version 1.2). 2024, [Online; https://github.com/reverseame/APOTHEOSIS]. [Accessed 2 August 2024].

[6] Radovanović M, Nanopoulos A, Ivanović M. Nearest neighbors in high-dimensional data: the emergence and influence of hubs. In: Proceedings of the 26th annual international conference on machine learning. New York, NY, USA: Association for Computing Machinery; 2009, p. 865–72.

[7] Malkov YA, Yashunin DA. Efficient and robust approximate nearest neighbor search using hierarchical navigable small world graphs. IEEE Trans Pattern Anal Mach Intell 2020;42(4):824–36.

[8] Malkov Y. HNSWlib. 2018, [Online; https://github.com/nmslib/hnswlib]. [Accessed 5 June 2023].

[9] Johnson J, Douze M, Jégou H. Billion-scale similarity search with GPUs. IEEE Trans Big Data 2019;7(3):535–47.

[10] Bartholomy. Hierarchical Navigable Small World: a scalable nearest neighbor search. 2023, [Online; https://github.com/brtholomy/hnsw]. [Accessed 5 June 2023].

[11] Morrison DR. PATRICIA–Practical Algorithm to Retrieve Information Coded in Alphanumeric. J ACM 1968;15:514–34.

[12] Liskov B, Zilles S. Programming with abstract data types. In: Proceedings of the ACM SIGPLAN symposium on very high level languages. New York, NY, USA: Association for Computing Machinery; 1974, p. 50–9.

[13] Gamma E, Helm R, Johnson R, Vlissides J. Design patterns: Elements of reusable object-oriented software. USA: Addison-Wesley Longman Publishing Co. Inc.; 1995.

[14] van Rossum G, Warsaw B, Coghlan A. PEP 8 – Style guide for Python code. 2013, [Online; https://peps.python.org/pep-0008/]. [Accessed 2 August 2024].