Comprehensive Survey Of Hashing Algorithms And Their Applications

Saeed M.Aldossari
Computer Science Department
Prince Sattam bin Abdulaziz University
Al-Kharj, Saudi Arabia
447540161@std.psau.edu.sa

Abdullah A. Sabr
Computer Science Department
Prince Sattam bin Abdulaziz University
Al-Kharj, Saudi Arabia
447540135@std.psau.edu.sa

Abdullah H. Almutairi
Computer Science Department
Prince Sattam bin Abdulaziz University
Al-Kharj, Saudi Arabia
447540061@std.psau.edu.sa

Mohamed O. Hegazi
Computer Science Department
Prince Sattam bin Abdulaziz University
Al-Kharj, Saudi Arabia
m.hegazi@psau.edu.sa

Abstract
Hashing has progressed from a simple constant-time lookup mechanism to a diverse set of algorithmic paradigms that underpin efficiency, security, and large-scale data management in modern computing. This survey organizes existing approaches into six principal categories: division-based hashing, dynamic hashing, cryptographic hashing, geometric and robust hashing, Bloom filter methods, and deep hashing. For each category, the paper outlines the key design principles, operational objectives, and characteristic performance trade-offs. The discussion connects these families of algorithms to their main application areas, including authentication, multimedia forensics, distributed storage, web systems, and approximate nearest-neighbour retrieval. Through comparative analysis, the survey emphasizes that the choice of hashing strategy is inherently context-dependent, shaped by constraints such as collision tolerance, memory efficiency, and semantic accuracy. Distinct from earlier reviews, this work brings together conventional and learning-based hashing techniques within a single analytical framework, highlighting the emergence of hybrid models that balance scalability, security, and similarity preservation in AI-driven and resource-limited environments.

Keywords:
Hashing algorithms, dynamic hashing, cryptographic hashing, Bloom filters, deep hashing, robust hashing.

1. INTRODUCTION

Hashing in general is a mechanism that converts variant size string into a fixed size string using a hash function. It is a fundamental operation that maps input objects from a large or unbounded domain to a limited address space through a deterministic transformation known as a hash function [1] [2].

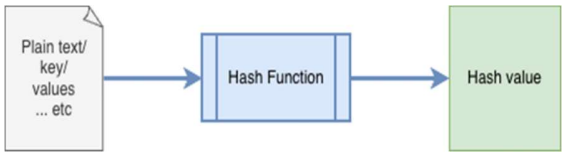

Figure 1 Hash function takes input and converts it to fixed-size string

It has been extensively adopted in performance-critical data processing pipelines due to its constant-time average-case lookup behavior and its ability to eliminate the need for ordered data structures or auxiliary indexing layers [3]. Beyond lookup acceleration, hashing also supports integrity verification, identity authentication, cache coordination, and content-based detection, thereby rendering it a multi-domain building block in data systems, communication networks, and security infrastructures [2] [4].

Historically, hashing techniques were conceived for static file organizations and compiler symbol tables. Subsequent developments introduced dynamic hashing to address scalability and collision growth in mutable datasets [3]. In parallel, cryptographic hash functions emerged as one-way constructs for password storage, message authentication, and digital signatures, where collision resistance and pre-image hardness rather than access latency are the primary objectives. In multimedia and computer vision, geometric and robust hashing methods have been devised to preserve similarity under transformations such as rotation, scaling, compression, or noise while still discriminating between dissimilar content [2] [1]. Probabilistic hashing, exemplified by Bloom filters, enables memory-efficient set membership tests at scale in distributed environments

Manuscript received November 5, 2025
Manuscript revised November 20, 2025
https://doi.org/10.22937/IJCSNS.2025.25.11.2

[2]. More recently, the proliferation of high-dimensional data retrieval has motivated deep hashing, wherein neural networks optimize binary embeddings to preserve semantic proximity in the Hamming space [2].

2. HASHING ALGORITHMS

A. The Division Method

The division method is a hashing technique where a key x is mapped to a hash table index using the formula:

$$H(x) = x \bmod m \quad (1)$$

If the table index starts at 1 instead of 0, the formula becomes:

$$H(x) = x \bmod m + 1. \quad (2)$$

The value of m must be chosen carefully, often as a prime number, to help distribute keys evenly and reduce collisions, especially when keys follow patterns. For example, if m = 10 and the key is 123456, the result is 6, meaning the key is stored at index 6. If m = 13 and the key is 987654, the result is 5, so the key is stored at index 5. However, this method can still cause clustering when many keys share similar structures (e.g., ending in the same digit), and performance may decline if keys are not well distributed [2].

B. Dynamic Hashing

Dynamic hashing is a set of techniques that let hash tables grow or shrink as data changes, overcoming the fixed-size limits of static hashing and maintaining fast lookup, insertion, and deletion. Approaches such as extendible hashing, linear hashing, and cuckoo hashing minimize collisions and avoid expensive full rehashing, keeping performance stable as key populations evolve. These methods are widely used in transactional databases, file systems, distributed networks, routing tables, and security systems where balanced data distribution and real-time responsiveness are essential. Their adaptability also makes them suitable for SDN (Software-Defined Networking) environments, were traffic patterns and data volume shift frequently. Current challenges include optimizing memory use, strengthening fault tolerance and security, and developing energy-efficient variants to meet the scalability demands of emerging domains such as IoT and AI [2] [3].

A typical example is extendible hashing, which stores keys in buckets referenced by a directory indexed by the low order bits of the hash value. Suppose the directory depth begins at $d = 1$, giving two buckets, 0 and 1. Inserting key K_1 (bits 00) places it in bucket 0. Inserting K_1 (bits 01) maps to the same bucket, causing overflow. Instead of rebuilding the table, only this bucket is split: its local depth becomes 2, the directory doubles to four entries (00, 01, 10, 11), and the affected keys are rehashed so that $K_1 \rightarrow 00$ and $K_2 \rightarrow 01$. Inserting a third key K_3 with bits 10 then maps directly to bucket 10 without further restructuring. This illustrates the core property of extendible hashing: only the overflowing bucket grows, preventing global rehashing and allowing incremental expansion [2].

C. Cryptographic Hashing

Cryptographic hashing is a one-way process that converts any input into a fixed-size digest used for security purposes such as password storage, digital signatures, and data integrity. Unlike dynamic hashing, which resizes tables for performance, cryptographic hashes focus on being irreversible and collision resistant. Algorithms like MD5, SHA-1, SHA-2, and SHA-3 take a message of any length and generate a unique output. For example:

SHA-1("Hello") = “f7ff9e8b7bb2e09b70935a5d785e0cc5d9d0abf0”

Even changing one letter produces a completely different hash, proving the avalanche effect [1] [2].

D. Geometric and Robust Hashing

Geometric hashing is mainly used in computer vision and computational geometry to match shapes or features even when objects are rotated, scaled, partially hidden, or noisy. Its key strength is efficiency, making it suitable for tasks like pattern recognition, 3D object matching, and protein structure alignment. It enables partial recognition by storing geometric relationships rather than raw data. Robust hashing, on the other hand, focuses on generating compact signatures that survive acceptable changes to the content. Instead of matching exact data, it preserves the “essence” of audio or images, allowing identification even after compression, resizing, or filtering. Robust audio hashing powers services like Shazam, while robust image hashing is

used in watermarking, fake-image detection, and copyright protection. Unlike cryptographic hashes, robust hashes remain similar when the content is slightly modified, making them ideal for multimedia authentication and retrieval [2].

A typical example is the classic geometric hashing pipeline. A model object is first represented by feature points. A pair of non-collinear points (p_i, p_j) is chosen as a basis, and all other points are transformed into this basis to produce invariant hash keys. These keys are inserted into a hash table. Later, when a query image is processed, the same basis-selection and hashing step is repeated. If many of the resulting keys vote for the same model object, the system declares a match—even if the object is scaled, rotated, or partially occluded. This ability to match transformed or incomplete data is what makes geometric hashing valuable in computer vision and bio-structure alignment [6]. On other hand, robust hashing, focuses on multimedia signals. Instead of hashing exact bytes, it produces a short signature that remains stable after acceptable modifications such as compression, resizing, or filtering. This allows systems to recognize the same content even when it is not bit-identical. For example, matching a recompressed image or a low-quality audio snippet. Robust audio hashing powers apps like Shazam, while robust image hashing is widely used in copyright tracking and fake-image detection. Unlike cryptographic hashes, robust hashes are designed to stay similar when the content is slightly modified, which makes them suitable for authentication and large-scale media search [7].

E. Bloom Filter Hashing

Bloom filters are space-efficient probabilistic data structures that enable constant-time membership queries while avoiding the storage cost of the full dataset. An item is processed by multiple independent hash functions, each of which maps it to a bit position in a fixed-size array. These positions are set to 1, and future lookups check the same positions to determine whether an element is possibly present or absent. The only source of error is the presence of false positives, whereas false negatives never occur; if a Bloom filter reports that an item is not stored, the result is guaranteed correct. [2] This behavior makes Bloom filters well suited for systems where memory efficiency and lookup speed are more important than perfect accuracy. To illustrate the mechanism, consider a bit array of length m =12, initialized to zero, and k =3 hash functions. Inserting two elements fills several bit positions, and although distinct items may set overlapping bits, the structure remains compact in constant time. When a third element is queried, the filter returnsdefinitely absent if any of the corresponding bit positions contain zero, and possibly present if all are set to one. The probability of a false positive after inserting n elements is commonly approximated by:

$$\mathrm{p} \approx \left(1 - \mathrm{e}^{-\frac{\mathrm{kn}}{\mathrm{m}}}\right)^{\mathrm{k}} \qquad (3)$$

Demonstrating that the error rate can be tuned by adjusting the number of hash functions and the bit-array size. Bloom filters are widely deployed in large-scale systems, including web caches that advertise stored URLs without transferring full lists, distributed databases such as Cassandra and HBase that avoid unnecessary disk reads, and networked systems that reduce message overhead in membership protocols. They are also used in spam detection, blockchain networks, and content delivery architectures where bandwidth and memory constraints are critical. Although classical Bloom filters remain dominant, more recent variants, such as Xor Filters and Binary Fuse Filters, improve memory usage and query throughput while maintaining the same probabilistic membership model [8][9].

F. Deep Hashing

Deep hashing is a learning-based hashing approach that uses deep neural networks to generate compact binary codes while preserving semantic similarity between data items. Unlike traditional hashing, which relies on hand-crafted features, deep hashing learns both the feature representation and the hash function jointly, allowing the model to optimise feature extraction and hash generation in a single end-to-end process. This makes it highly effective for large-scale similarity search, especially in domains where visual or semantic relationships cannot be captured by manually designed features [5]. A typical deep hashing system is trained on labelled or pairwise-related data so that samples belonging to the same class (e.g., two images of the same person) are mapped to nearby binary codes, while unrelated samples are pushed farther apart in Hamming space. After training, an

image may be converted into a 64-bit code such as: $b_1 = (1011010010111001 \ldots)$ and a visually similar image may produce: $b_2 = (1011010010111101 \ldots)$ differing by only a few bits. Because retrieval is performed using Hamming-distance lookup rather than high-dimensional floating-point comparison, the system can search millions of stored items in real time using only lightweight bit operations. This gives deep hashing a strong advantage in large-scale image retrieval, video deduplication, face recognition, cross-modal search (e.g., text–image matching), and recommendation engines where low latency and memory efficiency are essential

Compared with traditional hashing and conventional deep feature indexing, deep hashing offers three main benefits: First, it compresses high-dimensional features into short binary codes, reducing storage requirements. Second, it enables millisecond lookup time over massive datasets through Hamming search. Third, it scales gracefully as the size of the database grows, since code length remains fixed even when the number of samples increases. These properties have made deep hashing a key component in modern computer vision pipelines, multimedia databases, and AI-driven content retrieval systems [5].

3. APPLICATIONS OF HASHING ALGORITHMS

Hashing techniques are deployed across a wide spectrum of computing domains due to their ability to provide constant-time lookup, compact representation, tamper detection, and similarity preservation under varying constraints [3]. The choice of hashing method is dictated by operational objectives: classical and dynamic schemes priorities access latency and scalability; cryptographic hashing enforces integrity and trust; robust hashing preserves perceptual structure in multimedia pipelines; Bloom filters optimize probabilistic membership under memory constraints; and deep hashing enables efficient large-scale retrieval in high-dimensional semantic spaces. The following subsections survey the dominant application contexts aligned with these distinct hashing families.

A. Security and Integrity

Cryptographic hashing is central to authentication, integrity assurance, and non-repudiation. Unlike classical hashing, these functions are judged by collision resistance, pre-image hardness, and diffusion properties. Password storage subsystems employ salted and cost-amplified hashes such as bcrypt and Argon2 to withstand offline cracking. Message digests (SHA-256, SHA-3) secure software distribution, API signature validation, TLS negotiations, and audit logs. Blockchains append cryptographic hashes to enforce immutability of ledger states across untrusted nodes [1] [2].

B. Multimedia and Forensics

Robust and geometric hashing support content tracing under non-malicious distortions such as resizing, compression, cropping, or illumination differences. These techniques allow near-duplicate detection, copyright enforcement, broadcast monitoring, and tamper triage in forensic pipelines. In practice, services like YouTube content ID, Meta's image-matching pipelines, and stock-media registries employ perceptual hashes derived from DCT, SVD, or invariant feature transforms to flag reused or manipulated media at scale [2].

C. Networking and Distributed Systems

Hashing underpins routing, packet classification, and key distribution in distributed and software-defined networks. Dynamic hashing, Bloom filters, and robust hashing enable fast updates to SDN flow tables where rules change frequently, and latency budgets are strict. Scalable data stores and key–value systems such as Cassandra, HBase, and Dynamo-style architectures rely on consistent or dynamic hashing to distribute keys evenly and avoid global rebalancing as nodes join or leave. Bloom filters minimize lookup cost and suppress redundant traffic in web caching, large-scale databases, P2P overlays, and WAN optimization appliances, while robust hashing supports content-based routing and multimedia identification even after compression or transformation. Emerging AI-driven networks further adopt deep hashing to index high-dimensional telemetry or sensor feeds, allowing similarity search at line rate without inflating memory or CPU cost [4] [5].

D. Web Systems and Search Infrastructures

In web systems, hashing enables fast indexing, duplicate detection, and load-balanced caching. Bloom filters are widely deployed to compress membership sets in collaborative web caches, where each cache

broadcasts only hashed summaries instead of full URL lists. In free-text and document search, Bloom filters support rapid term existence checks, reducing I/O in large, inverted indexes. Web-scale clustering of pages similarly uses resemblance hashing to detect near-duplicate documents in crawl pipelines [2] [4].

E. Approximate Nearest Neighbour Retrieval

Approximate nearest neighbour retrieval relies on hashing methods that compress high-dimensional embeddings into compact binary codes while preserving similarity structure. Deep hashing enables sub-linear search over billion-scale corpora, making them integral to vision retrieval, face recognition, recommender systems, and cross-modal alignment. Large-scale platforms such as Facebook'sFAISS and Google's internal retrieval stacks use learned hash codes to perform fast embedding lookups with minimal loss of semantic accuracy. Similar techniques drive e-commerce ranking engines, where strict latency budgets require rapid candidate generation before full-precision re-scoring [5].

4. SUMMARY AND CONCLUSION

Hashing has evolved from a constant-time lookup tool into a versatile set of algorithms serving diverse computational needs. Traditional and dynamic hashing continue to form the foundation of storage engines and transactional systems by ensuring predictable access and scalability. Cryptographic hashing emphasizes irreversibility and collision resistance, providing the backbone for authentication, integrity verification, and tamper-evident ledgers. Geometric and robust hashing extend these ideas to perceptual domains, enabling resilience to transformations such as rotation, scaling, and compression—key to multimedia tracing and large-scale content identification. Probabilistic approaches, particularly Bloom filters, trade exactness for memory and bandwidth efficiency in distributed and caching systems. Finally, deep hashing introduces learning-based embeddings that preserve semantic similarity for large-scale retrieval across visual and multimodal data. Each hashing family addresses a distinct objective—speed, security, invariance, efficiency, or semantic retrieval—and no single method dominates across all domains. The growing trend is toward hybrid approaches that merge these strengths, combining cryptographic assurance with similarity awareness and probabilistic efficiency. Hashing thus remains a compact, computation-light abstraction central to scalable data systems, secure infrastructures, and AI-driven environments.

Table 1: Hashing Algorithms and Their Suitable Applications

Application Domain	Most Suitable Hashing Algorithms	Justification
Databases & storage	Classical / Dynamic hashing: maps keys to buckets with constant-time access, with dynamic variants allowing table growth without full rehashing.	Lookup latency and mutation tolerance dominate over security or perceptual invariance. Guarantees O(1)average access; extendible/linear hashing avoid global rebuilds.
Security & integrity	Cryptographic hashing (SHA-2/3, bcrypt, Argon2): produces one-way collision-resistant digests for trust and authentication.	The domain is adversarial; integrity and non-invertibility are mandatory. Salted / slow hashes mitigate brute-force; tamper evident logs and signatures.
Multimedia & forensics	Robust / geometric hashing generates similarity-preserving signatures stable under scaling, cropping, compression or noise.	Must detect near-duplicates and tampering even after benign transformations. Used in ContentID, watermarking, forensic pipelines; built on DCT/SVD/invariants.
Networking & distributed systems	Dynamic hashing & Bloom filters: scalable placement and probabilistic membership	State must fit in constrained routers and adapt to churn with minimal delay. Enables consistent key spread; BF suppresses negatives

	for routing, DHTs and SDN tables.	without storing keys.
Web systems & caching	Bloom filters + classical maps: fast membership tests and URL/CDN indexing with minimal memory footprint.	Prevents costly cache/disk lookups and reduces crawler/network load at scale. Used in Redis, Memcached, CDN front-end, search indexing stacks.
Approximate NN / retrieval	LSH / Deep hashing: projects high-dimensional features to binary codes that preserve semantic closeness in Hamming space.	Enables sublinear nearest-neighbour search over billion-scale corpora.

REFERENCES

[1]Kale, A. M., & Dhamdhere, S. (2018). Survey paper on different type of hashing algorithm. International Journal of Advance Scientific Research Algorithm, 3(2).

[2] Singh, M., & Garg, D. (2009, March). Choosing best hashing strategies and hash functions. In 2009 IEEE International Advance Computing Conference (pp. 50-55). IEEE.

[3] Lafta, N. A., & Al-fiskhaltom, F. R. F. (2023). A Comprehensive Survey of Dynamic Hashing Techniques in Network Data Processing. Babylonian Journal of Networking, 2023, 89-93.

[4] Patra, S. P., & Rani, M. (2025). Evaluation and Categorization of Hashing Algorithms Based on Their Applications. IAENG International Journal of Applied Mathematics, 55(3).

[5] Luo, X., Wang, H., Wu, D., Chen, C., Deng, M., Huang, J., & Hua, X. S. (2023). A survey on deep hashing methods. ACM Transactions on Knowledge Discovery from Data, 17(1), 1-50.

[6] M. Coluzzi, A. Brocco, and T. Leidi, “A survey and fair comparison of consistent hashing”, in CEUR Workshop Proceedings, vol. 3478, 2023.

[7] L. Du, A. T. S. Ho, and R. Cong, “Perceptual hashing for image authentication: A survey”, Signal Processing: Image Communication, vol. 81, p. 115713, Feb. 2020.

[8] T. Graf and D. Lemire, “Xor filters: Faster and smaller than Bloom and Cuckoo filters”, ACM Journal of Experimental Algorithmics, vol. 27, pp. 1–16, 2022.

[9] T. Graf and D. Lemire, “Binary Fuse Filters: Fast and smaller than Xor Filters”, ACM Journal of Experimental Algorithmics, vol. 27, pp. 1–17, 2022.