PURESTORAGE®

WHITE PAPER

The Storage Reality of SQL Server Vector Search

Overcoming I/O and capacity challenges to deliver enterprise AI performance

Contents

Introduction 3

SQL Server 2025 vector search 3

- How DiskANN works 4
- Working with vector embeddings in SQL Server 4
- Challenges 8

How Pure Storage FlashArray solves these challenges 10

- Validation environment 10
- How FlashArray addresses vector search storage demands 11
- Enabling workload consolidation 13

Conclusion 14

Introduction

Microsoft SQL Server 2025 introduces native vector search capabilities, enabling similarity queries and retrieval-augmented generation (RAG) workloads to run directly within the database engine. For organizations already standardized on SQL Server, this consolidates AI-related data operations onto existing infrastructure and eliminates the need for a separate vector database.

The storage implications are significant. Vector embeddings expand dataset footprints substantially, the I/O profile differs from traditional database workloads, and hybrid environments introduce contention between vector operations and transactional queries. Storage infrastructure that was adequate for conventional SQL Server deployments may not deliver the latency characteristics that vector search requires.

This paper examines these challenges and presents validation data with a 100-million-vector dataset, demonstrating that purpose-built flash storage can address the I/O and capacity demands of enterprise-scale vector search.

SQL Server 2025 vector search

Vector search enables queries based on semantic similarity rather than exact matching. Instead of asking "find all products where category = 'electronics,'" a vector query asks, "find products most similar to this description." The input and the searchable data are both represented as vectors: arrays of floating-point numbers that encode meaning in high-dimensional space.

This capability underpins a range of AI applications:

- RAG systems retrieve relevant context to ground large language model responses.
- Recommendation engines find similar products or content.
- Search systems return semantically related results even when keywords don't match exactly.

SQL Server 2025 brings this functionality into the database engine through the VECTOR data type and several vector search related functions. Developers can store embeddings alongside relational data, query them with T-SQL, and apply enterprise-grade SQL Server security, backup, and recovery mechanisms.

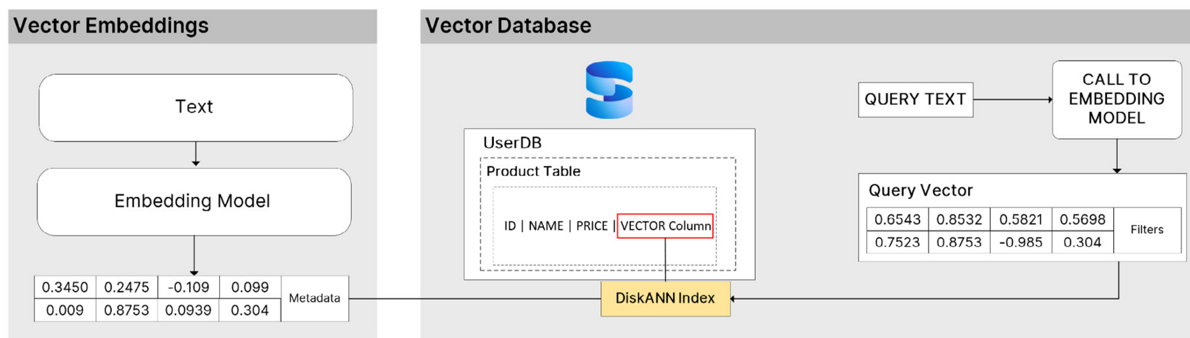

FIGURE 1 Vector embedding and search high level workflow

How DiskANN works

To execute similarity searches across millions of vectors without comparing every one, SQL Server 2025 uses DiskANN (disk-based approximate nearest neighbor), an algorithm developed by Microsoft Research.

DiskANN's key design choice is keeping only a compressed navigational structure in memory while the full-precision vectors remain in storage. This makes vector search feasible without requiring the entire index to fit in RAM: a 100-million-vector dataset that would need hundreds of gigabytes of memory under an in-memory approach can run with under 50GB per node. The trade-off is that query performance now depends directly on how fast the storage can deliver random reads.

Comparison with in-memory approaches

Other vector databases commonly use HNSW (hierarchical navigable small world), an algorithm that keeps the entire index in memory. HNSW delivers consistent low-latency queries because all data access occurs in RAM, but memory requirements scale linearly with dataset size. For enterprise-scale deployments, this becomes cost-prohibitive.

Characteristic	HNSW (in memory)	DiskANN (SQL Server 2025)
Index location	Entirely in RAM	Compressed graph in RAM, vectors on storage
Memory requirement	Full dataset size	20–30% of dataset size
Storage I/O during query	Minimal	High (random reads per query)
Latency determinant	Memory bandwidth	Storage latency
Cost profile	High memory cost	Lower memory, storage-dependent

TABLE 1 HNSW vs. DiskANN characteristics

DiskANN shifts the infrastructure constraint from memory capacity to storage capability. This is a deliberate engineering trade-off, not a limitation, but it changes what the storage layer must deliver.

Working with vector embeddings in SQL Server

The VECTOR data type stores fixed-dimension arrays of numerical values representing embeddings. When defining a vector column, you specify the number of dimensions to match your embedding model's output.

Creating a table with vector storage

Use the below code to create a table with vector storage.

```
CREATE TABLE Products (
    ProductID         INT PRIMARY KEY,
    ProductName       NVARCHAR(255),
    Description       NVARCHAR(MAX),
    Category          NVARCHAR(100),
    Price             DECIMAL(10,2),
    DescriptionVector VECTOR(1536)  -- Matches your embedding model's dimensions
);
```

Inserting vector data

Embeddings are typically generated by an external model and inserted as arrays. SQL Server accepts vectors in JSON array format:

```
INSERT INTO Products (ProductID, ProductName, Description, Category, Price, DescriptionVector)
VALUES (
    1,
    'Wireless Headphones',
    'Noise-canceling over-ear headphones with 30-hour battery life',
    'Electronics',
    199.99,
    CAST('[0.0123, -0.0456, 0.0789, ...]' AS VECTOR(1536))
);
```

In practice, embeddings can be generated externally or within SQL Server itself. For external generation, application code calls an embedding API and passes the result as a parameter:

```
-- Typical pattern: application generates embedding, passes as parameter
INSERT INTO Products (ProductID, ProductName, Description, Category, Price, DescriptionVector)
VALUES (@ProductID, @ProductName, @Description, @Category, @Price, AI_GENERATE_EMBEDDINGS(Description)
);
```

SQL Server 2025 also supports native embedding generation using AI_GENERATE_EMBEDDINGS(), which calls a configured model endpoint directly from T-SQL:

```
-- Native generation: SQL Server generates embedding from text
UPDATE Products
SET DescriptionVector = AI_GENERATE_EMBEDDINGS(Description)
WHERE DescriptionVector IS NULL;
```

Creating a vector index

Vector search can operate in two modes. Exact search (KNN) performs a full table scan of all rows' vectors, guaranteeing the best possible results but requiring compute time that scales linearly with dataset size. At millions of vectors, this becomes impractical for interactive applications.

For smaller datasets or when precision is critical, you can query vectors directly using VECTOR_DISTANCE without an index:

```
DECLARE @QueryVector VECTOR(1536) = CAST('[0.0234, -0.0567, 0.0891, ...]' AS VECTOR(1536));
SELECT TOP 10
    ProductID,
    ProductName,
    VECTOR_DISTANCE('cosine', DescriptionVector, @QueryVector) AS distance
FROM Products
ORDER BY distance;
```

Approximate search (ANN) uses an index structure to narrow the search space, examining only a subset of candidates. This trades a small accuracy reduction for dramatically faster response times. In practice, a well-tuned ANN index returns the correct top results in the vast majority of queries while completing in milliseconds rather than seconds.

SQL Server 2025 implements ANN search through DiskANN indexes. Without an index, vector searches fall back to exact KNN scan.

```
CREATE VECTOR INDEX IX_Products_DescriptionVector
ON Products(DescriptionVector)
WITH (
    METRIC = 'cosine',
    TYPE = 'DiskANN',
    MAXDOP = 4
);
```

Note: The parameters shown here are illustrative. See the Microsoft documentation for the full list of index options and tuning guidance.

Executing a vector search

The VECTOR_SEARCH function returns the most similar records to a query vector:

```
DECLARE @QueryVector VECTOR(1536) = CAST('[0.0234, -0.0567, 0.0891, ...]' AS VECTOR(1536));
-- This uses the DiskANN index for fast approximate
SELECT
    p.ProductID,
    p.ProductName,
    p.Price,
    v.distance
    FROM VECTOR_SEARCH(
    TABLE = Products AS p,
    COLUMN = DescriptionVector,
    SIMILAR_TO = @QueryVector,
    METRIC = 'cosine',
    TOP_N = 10
) AS v
ORDER BY v.distance;
```

For hybrid queries that combine vector similarity with traditional filters, like a WHERE clause:

```
SELECT TOP 10
    p.ProductID,
    p.ProductName,
    p.Price,
    V.distance
FROM VECTOR_SEARCH(
    TABLE = Products AS p,
    COLUMN = DescriptionVector,
    SIMILAR_TO = @QueryVector,
    METRIC = 'cosine',
    TOP_N = 50
) AS v
WHERE p.Category = 'Electronics'  -- Filter the candidates
  AND p.Price < 500.00
ORDER BY v.distance;
```

Challenges

The DiskANN architecture introduces three categories of infrastructure challenge: I/O profile, storage capacity, and workload contention.

I/O profile

Each vector similarity query generates multiple storage read operations as DiskANN traverses the index graph and retrieves candidate vectors. These reads have specific characteristics:

- **Small to moderate block sizes:** During testing, observed read sizes ranged from 4KB to 32KB. Actual I/O patterns depend on query optimizer decisions, read-ahead behavior, and index structure.
- **Random access patterns:** During graph traversal, the search algorithm jumps between nodes based on similarity, resulting in access patterns that are effectively random from the storage system's perspective. Once the DiskANN index narrows to candidate regions, vectors within those regions may be physically adjacent.
- **Latency sensitivity:** Vector search often serves interactive applications. Users expect sub-second responses. Each storage read contributes directly to query latency.

This profile is challenging for storage architectures optimized for different workloads. Sequential read optimization provides no benefit during graph traversal. Read-ahead caching is ineffective when the next read location is unpredictable. Storage systems designed for large-block throughput may deliver high bandwidth numbers while failing to sustain the IOPS and latency required for vector search.

Storage latency during these operations directly determined query response time.

The capacity problem (storage bloat)

Vector embeddings consume substantial storage. A single embedding from a model like OpenAI's text-embedding-ada-002 contains 1,536 floating-point numbers. At 4 bytes per float, each embedding is approximately 6KB before any overhead.

Dataset size	Embedding dimensions	Raw vector storage	With index overhead
1 million	1,536	~6GB	~10–15GB
10 million	1,536	~60GB	~100–150GB
100 million	1,536	~600GB	~1–1.5TB

TABLE 2 Dataset storage requirements

Note: These figures represent the vectors alone. Production deployments also store the original data, metadata, and potentially multiple embedding columns for different models or use cases. Storage requirements grow quickly.

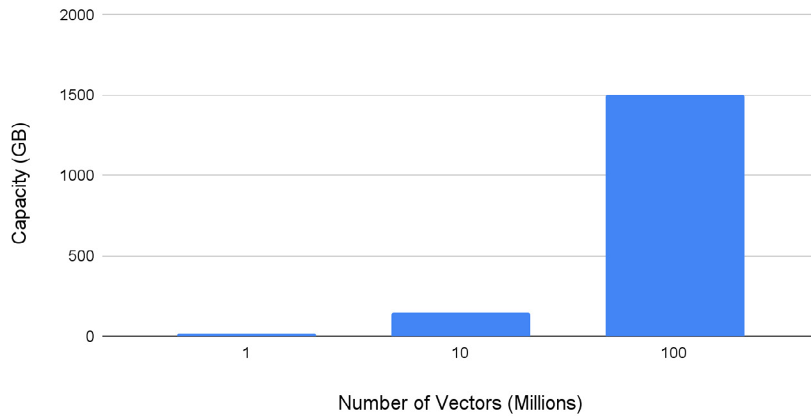

FIGURE 2 Physical storage footprint scaling with number of vectors

Vector data also resists traditional compression. Embeddings are normalized floating-point arrays with high entropy. Standard compression algorithms that rely on pattern recognition find little to exploit. Without storage-level data reduction, organizations pay the full capacity cost.

Workload contention

Enterprises rarely deploy dedicated infrastructure for vector search. More commonly, they enable vector capabilities on existing SQL Server instances that also serve transactional workloads, like ERP systems, customer databases, and financial applications.

This creates resource contention at the storage layer. A spike in vector search activity generates a surge of random I/O. If the storage system cannot absorb this load while maintaining service levels for other workloads, transactional query latency suffers.

The buffer pool compounds this problem. SQL Server uses the buffer pool to cache frequently accessed data pages. Vector search operations can pull large volumes of data into the pool, displacing pages needed by transactional workloads and forcing additional storage reads. Without workload isolation, vector search can degrade the performance of unrelated applications sharing the same instance.

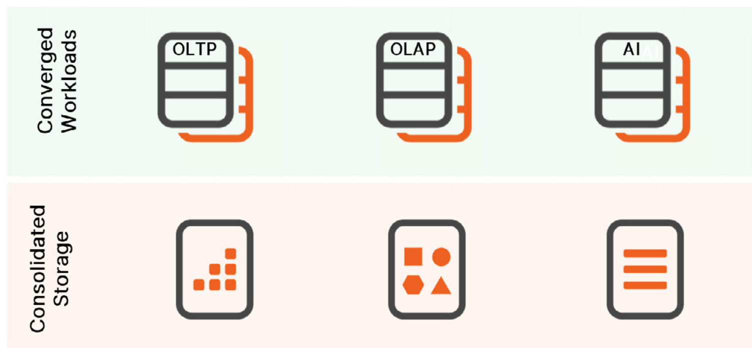

FIGURE 3 Workload consolidation on consolidated storage

How Pure Storage FlashArray solves these challenges

Pure Storage® FlashArray™ is designed for the I/O patterns, capacity demands, and mixed-workload requirements that SQL Server vector search creates. This section covers the architecture, how it addresses each challenge, and validation data from testing at scale.

FlashArray is the Pure Storage block and file storage platform, available in models that scale from departmental workloads to large enterprise deployments.

- **FlashArray//C™:** Cost-optimized, capacity-focused workloads
- **FlashArray//X™:** General-purpose enterprise workloads
- **FlashArray//XL™:** Performance-intensive, consolidated, large-scale deployments

All models share the Purity operating environment and core architecture. The capabilities described below apply across the family, with the FlashArray//XL used for the validation testing due to its performance headroom at scale.

Validation environment

To quantify how FlashArray tackles the challenges of SQL Server vector search, tests were conducted using a 100-million-vector dataset and the following parameters:

Dataset configuration: The full 100-million-vector dataset and index were replicated onto eight separate SQL Server nodes. To maximize concurrency and prevent database locking bottlenecks, the data was divided on each node into 10 tables (10 million vectors each).

Methodology: To simulate a real-world traffic storm, four independent client machines generated search queries across all 10 tables on all eight nodes. We segmented the workload to minimize contention, and each client was assigned a dedicated group of tables (e.g., Client 1 targeted Tables 1 to 3) rather than accessing the entire dataset randomly. We ran these workloads in sustained 30-second bursts, systematically increasing the number of concurrent users to find the system's peak limits.

Component	Embedding dimensions
SQL Server instances	8
Vectors per instance	100 million (10 tables × 10 million vectors)
Total vectors	100 million/instance
Total dataset size	~470GB per instance (data + index)
Vector dimensions	768
Memory per node	45GB (less than 10% of dataset size)
Storage	Pure Storage FlashArray//XL

TABLE 3 Test components

Memory and storage constraints: The memory configuration was intentional; limiting each node to 45GB (against an ~470GB dataset) forced the DiskANN index to retrieve vectors from storage rather than memory. This strictly validated the storage array's capability to handle the random read profile of vector traversal.

Note: Results reflect this specific test environment. Production performance varies with dataset characteristics, query patterns, and infrastructure configuration. Proof-of-concept testing is recommended for capacity planning.

This GitHub repository contains the benchmarking harness used to validate SQL Server 2025 vector search performance on Pure Storage FlashArray. Follow the steps below to build the load generator and reproduce the segmented traffic patterns described in this white paper.

How FlashArray addresses vector search storage demands

The test configuration above creates a deliberately storage-intensive workload. With vectors forced to disk, three characteristics of the I/O profile become critical for performance.

Sustaining the I/O profile

Challenge: DiskANN generates small-block (18–25KB) random reads at high volume. Storage must deliver consistent sub-millisecond latency under this pattern.

Solution: FlashArray DirectFlash® architecture connects the storage controller directly to raw NAND flash, bypassing the translation layers present in commodity SSDs. This eliminates latency variability and allows the array to optimize I/O scheduling for the actual access patterns it receives.

Validation: Testing with 100 million vectors across 8 SQL Server nodes demonstrated:

Databases	Queries per second	Average storage latency
1	~1,350	0.24ms
2	~2,700	0.23ms
4	~5,500	0.25ms
8	~11,500	0.27ms

TABLE 4 Small-block testing results

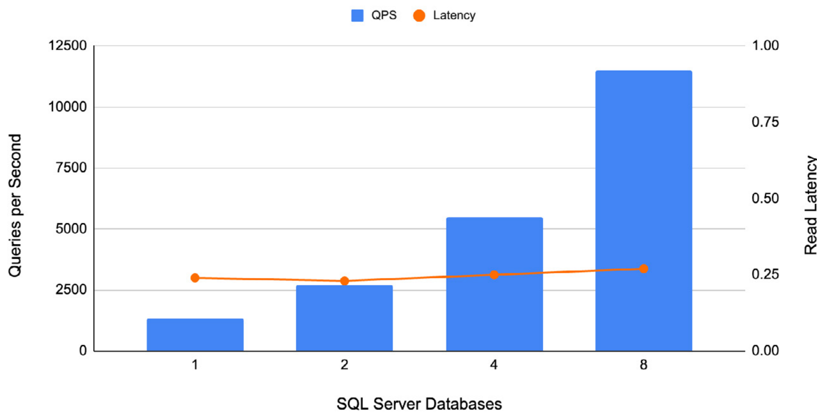

FIGURE 4 Scaling analysis of consolidated storage for vector embedding retrieval

The validation testing confirmed that FlashArray delivers the exceptional storage performance required to decouple vector search from expensive memory resources, allowing SQL Server 2025 to scale efficiently.

- **Exceptional sub-millisecond storage performance:** Regardless of the increasing throughput, storage latency remained consistently ultra-low, hovering between 0.23ms and 0.29ms. This proves that FlashArray can sustain sub-millisecond response times even while handling the difficult, random, small-block I/O traffic (18–25 KB) typical of DiskANN graph traversals.
- **Massive scalability with minimal host memory:** By offloading the vector data to efficient storage, we successfully reduced the memory footprint per node to just 45GB (less than 10% of the dataset size). This confirms that users can scale to larger datasets, from 100 million to 1 billion+ vectors without the prohibitive cost of ensuring every node has enough RAM to hold the entire index.
- **Near-linear throughput growth:** As the cluster expanded to 8 nodes, queries per second (QPS) increased by nearly 8.5x (reaching ~11,500 QPS).

Reducing the storage footprint

Challenge: Vector embeddings expand storage requirements substantially. A 100-million-vector dataset with 1,024 dimensions requires over 1.8TB after indexing overhead.

Solution: FlashArray applies data reduction inline as data is written, using pattern-matching deduplication and compression. While vector data has high entropy, the algorithms find optimization opportunities that traditional compression misses.

Validation: Testing with the 100-million-vector dataset showed:

Metric	Value
Logical data footprint	~1.8TB
Dimension and number of vectors	1,024, ~247 million vectors
Physical data footprint	~630GB
Data reduction ratio	2.7:1
Capacity savings	~63%

TABLE 5 Testing results

The 2.7:1 reduction ratio reduces effective storage costs significantly. This applies to both vector columns and DiskANN index structures.

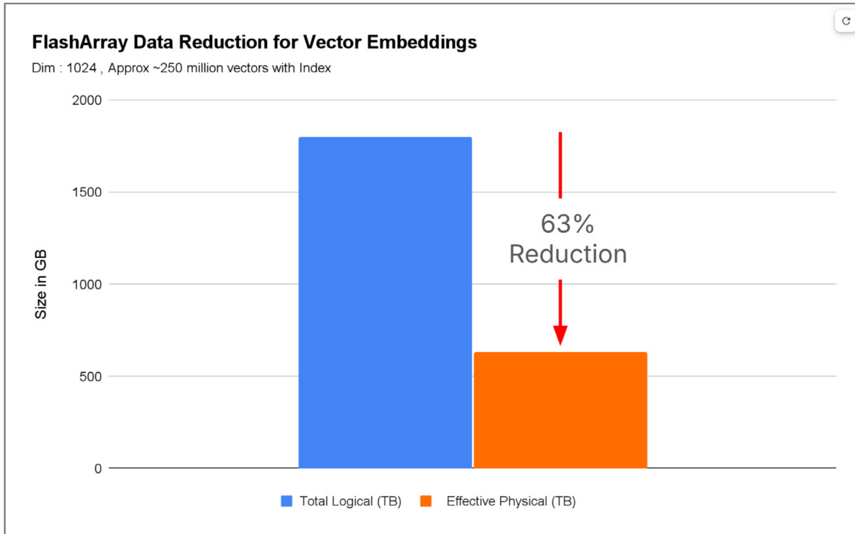

FIGURE 5 Vector embedding storage footprint before and after data reduction

Note: Results will vary based on embedding dimensions, storage layouts, dataset type, and indexing parameters and tunables. We strongly advise running a proof of concept (PoC) to validate performance for your specific use case.

Enabling workload consolidation

Challenge: Vector search shares infrastructure with transactional workloads. A spike in similarity queries can starve OLTP applications of I/O.

Solution: FlashArray includes quality-of-service controls that prevent any single workload from monopolizing array resources. When vector search activity spikes, transactional workloads continue to receive consistent latency.

This enables consolidation rather than isolation. Organizations can run vector search on the same FlashArray volumes as existing SQL Server databases without building separate infrastructure.

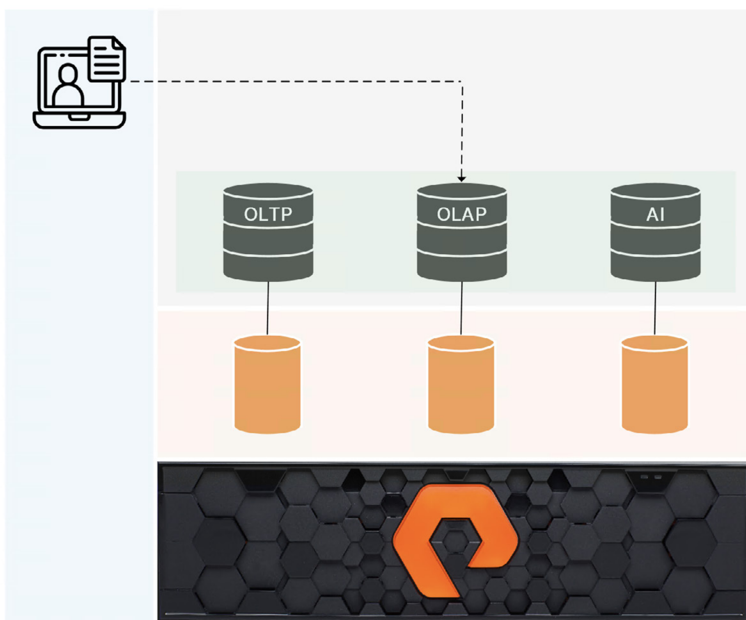

FIGURE 6 Consolidated database workloads on centralized storage on FlashArray

Conclusion

SQL Server 2025 vector search brings AI capabilities to the enterprise database, but the underlying DiskANN architecture creates infrastructure requirements that differ from traditional database workloads. Storage becomes the critical path for query performance, vector embeddings expand capacity requirements substantially, and mixed workloads introduce contention risks.

Pure Storage FlashArray addresses these requirements directly. DirectFlash architecture delivers the consistent low-latency random read performance that DiskANN demands. Inline data reduction offsets capacity expansion. Quality-of-service controls enable vector and transactional workloads to coexist on shared infrastructure.

The validation results demonstrate that these capabilities translate to measurable outcomes: near-linear throughput scaling, sub-millisecond latency at scale, and meaningful capacity savings on vector data.

For organizations planning SQL Server vector search deployments, storage infrastructure is not a secondary consideration. It is a primary factor in whether the deployment meets performance requirements. FlashArray provides a foundation designed for the workload.

Try out the SQL Server Test Drive.

©2026 Pure Storage, the Pure Storage P Logo, DirectFlash, FlashArray, FlashArray//C, FlashArray//X, FlashArray//XL, and the marks in the Pure Storage Trademark List are trademarks or registered trademarks of Pure Storage Inc. in the U.S. and/or other countries. The Trademark List can be found at purestorage.com/trademarks. Other names may be trademarks of their respective owners.

ID13896-01-en-01/26