VectorizationProvider

|\_\_\_\_\_\_\_\_\_\_\_\_\_\_> VectorSupportUtil : Top level class for various vectorized algorithms. Inherited by PanamaVectorizationProvider, NativeVectorizationProvider and DefaultVectorizationProvider.

|\_\_\_\_\_\_\_\_\_\_\_\_\_\_> VectorTypeSupport: Top level backing storage class, inherited by OffHeapVectorProvider and ArrayVectorProvider. Former uses Panama MemorySegments while later allocates primitive arrays as the backing storage.

A screenshot of a computer code

Description automatically generated

A computer screen shot of a program code

Description automatically generated

A blue text on a white background

Description automatically generated

A computer screen shot of a computer screen

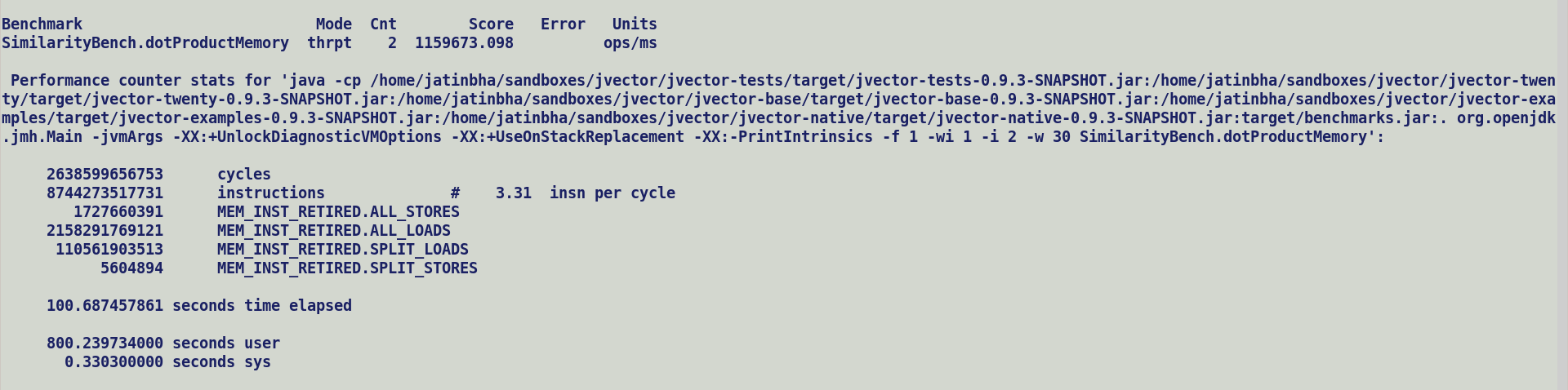
Description automatically generated

A close-up of a computer screen

Description automatically generated

A close-up of a computer screen

Description automatically generated

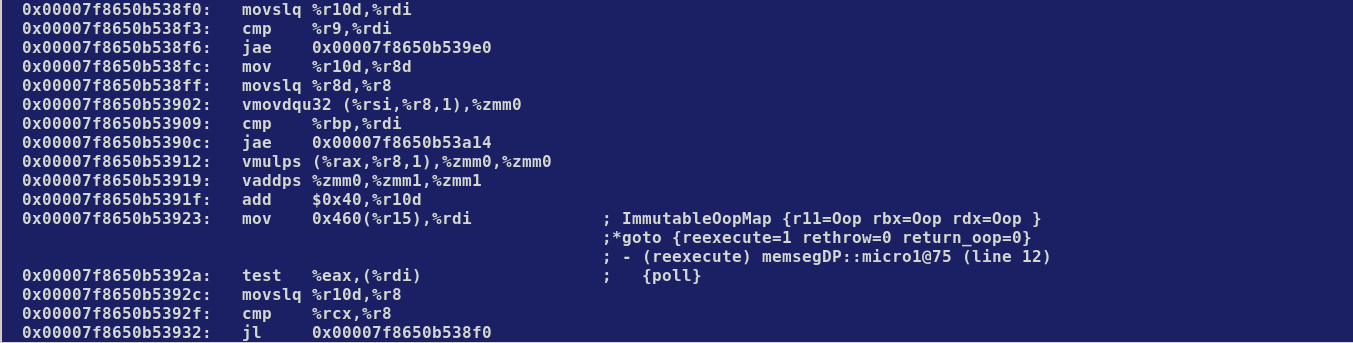


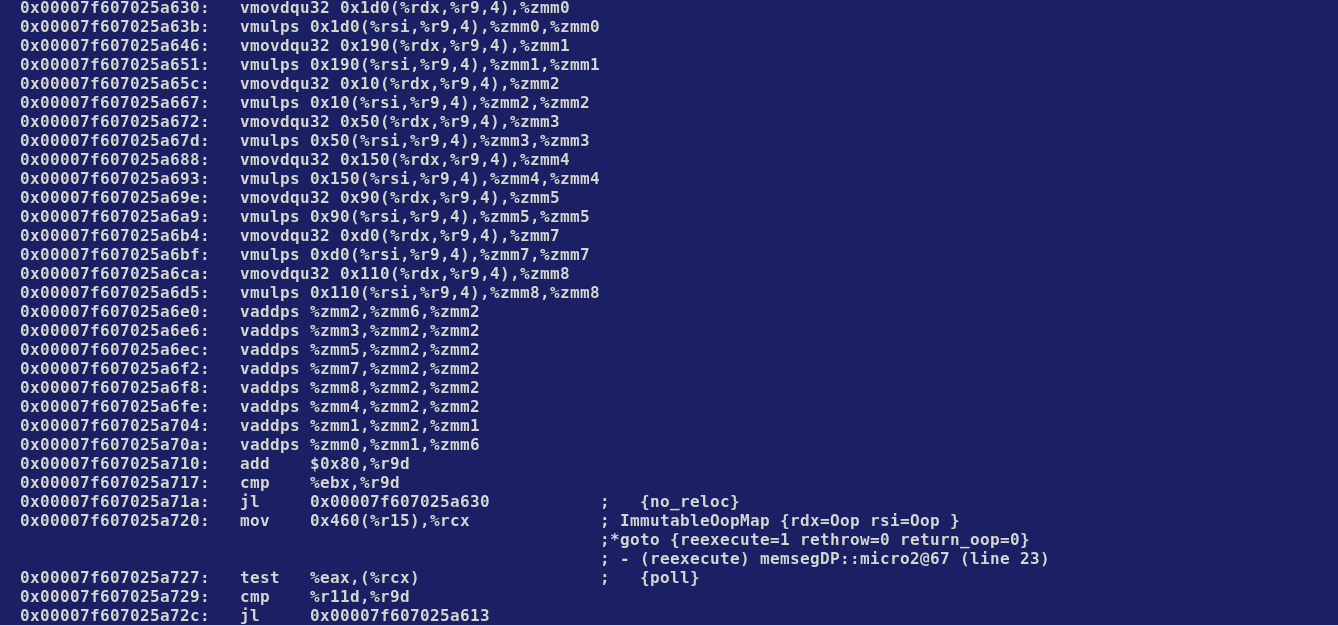
A screenshot of a computer program

Description automatically generated

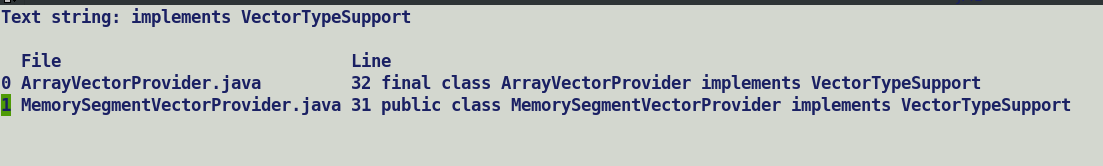
A close-up of a computer screen

Description automatically generated





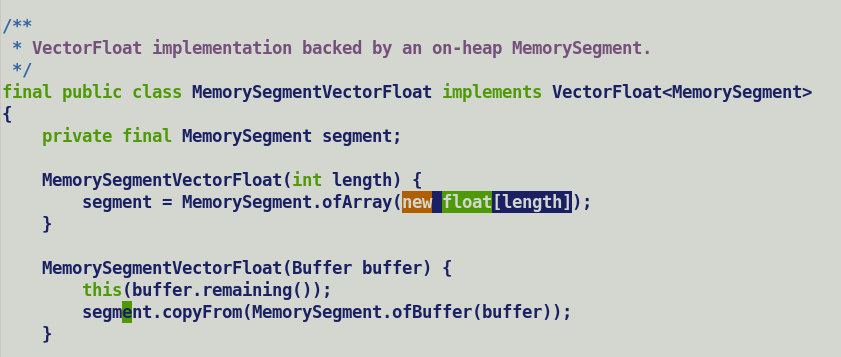




A close-up of a computer screen

Description automatically generated

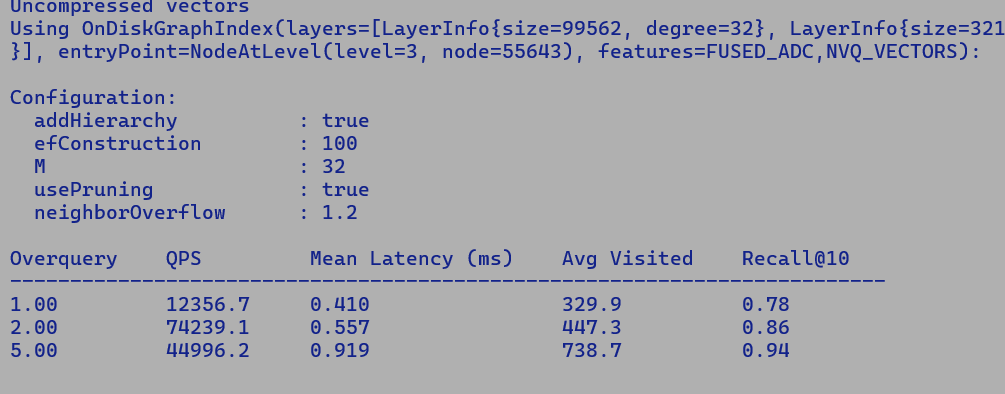
Array backed memory segments are not align able. Alignment can be enforced only on off-help segments.



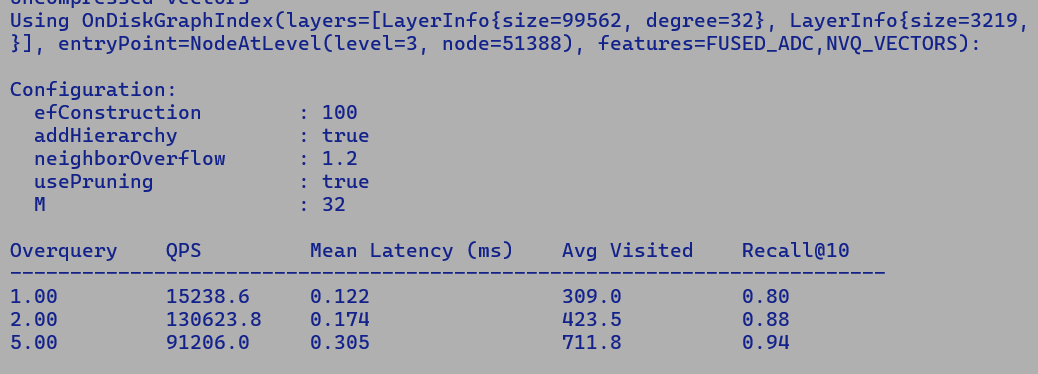
TODO:

1. Run benchmarks with MaxVectorSize=256
2. Check performance on Sierra Forest.

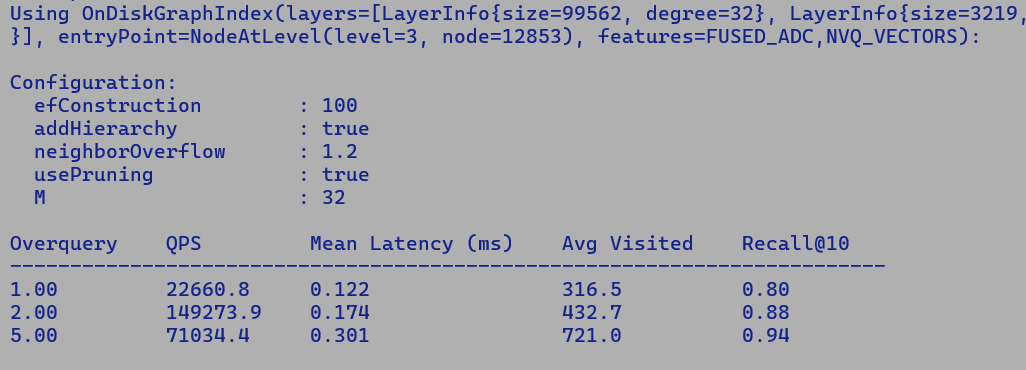
DefaultVectorizationProvider



NativeVectorizationProvider



PanamaVectorizationProvider



**ADC – Quick ADC – Quicker/Fused ADC**

Let's revisit the computation of asymmetric distance for each database vector in the ADC example to ensure accuracy. We'll clarify the process and correct any discrepancies in the sub-vector comparisons.

**Example Setup Recap**

**Original Database Vectors**

* **Vectors:**
  + ( \mathbf{v}\_1 = [1, 2, 3, 4] )
  + ( \mathbf{v}\_2 = [5, 6, 7, 8] )
  + ( \mathbf{v}\_3 = [9, 10, 11, 12] )
  + ( \mathbf{v}\_4 = [13, 14, 15, 16] )

**Splitting into Sub-vectors**

Each vector is split into two sub-vectors:

* ( \mathbf{v}\_1 = [1, 2] ) and ( [3, 4] )
* ( \mathbf{v}\_2 = [5, 6] ) and ( [7, 8] )
* ( \mathbf{v}\_3 = [9, 10] ) and ( [11, 12] )
* ( \mathbf{v}\_4 = [13, 14] ) and ( [15, 16] )

**Codebooks**

1. **Codebook for First Sub-vector:**
   * Centroids: ( { [1, 2], [5, 6], [9, 10], [13, 14] } )
2. **Codebook for Second Sub-vector:**
   * Centroids: ( { [3, 4], [7, 8], [11, 12], [15, 16] } )

**Query Processing with ADC**

**Query Vector**

* **Query Vector:**
  + ( \mathbf{q} = [2, 3, 4, 5] )
  + Split into sub-vectors: ( \mathbf{q}\_1 = [2, 3] ) and ( \mathbf{q}\_2 = [4, 5] )

**Distance Computation**

1. **Compute Distances for First Sub-vector (( \mathbf{q}\_1 = [2, 3] )):**
   * Compare ( \mathbf{q}\_1 ) against each centroid in the first codebook:
     + Distance to ( [1, 2] ): [ \text{dist}([2, 3], [1, 2]) = \sqrt{(2-1)^2 + (3-2)^2} = \sqrt{1 + 1} = \sqrt{2} ]
     + Distance to ( [5, 6] ): [ \text{dist}([2, 3], [5, 6]) = \sqrt{(2-5)^2 + (3-6)^2} = \sqrt{9 + 9} = \sqrt{18} ]
     + Distance to ( [9, 10] ): [ \text{dist}([2, 3], [9, 10]) = \sqrt{(2-9)^2 + (3-10)^2} = \sqrt{49 + 49} = \sqrt{98} ]
     + Distance to ( [13, 14] ): [ \text{dist}([2, 3], [13, 14]) = \sqrt{(2-13)^2 + (3-14)^2} = \sqrt{121 + 121} = \sqrt{242} ]
2. **Compute Distances for Second Sub-vector (( \mathbf{q}\_2 = [4, 5] )):**
   * Compare ( \mathbf{q}\_2 ) against each centroid in the second codebook:
     + Distance to ( [3, 4] ): [ \text{dist}([4, 5], [3, 4]) = \sqrt{(4-3)^2 + (5-4)^2} = \sqrt{1 + 1} = \sqrt{2} ]
     + Distance to ( [7, 8] ): [ \text{dist}([4, 5], [7, 8]) = \sqrt{(4-7)^2 + (5-8)^2} = \sqrt{9 + 9} = \sqrt{18} ]
     + Distance to ( [11, 12] ): [ \text{dist}([4, 5], [11, 12]) = \sqrt{(4-11)^2 + (5-12)^2} = \sqrt{49 + 49} = \sqrt{98} ]
     + Distance to ( [15, 16] ): [ \text{dist}([4, 5], [15, 16]) = \sqrt{(4-15)^2 + (5-16)^2} = \sqrt{121 + 121} = \sqrt{242} ]

**Compute Asymmetric Distance for Each Database Vector**

* **For ( \mathbf{v}\_1 ):**
  + Quantized indices: ( [0, 0] )
  + Asymmetric distance: [ \text{ADC}(\mathbf{v}\_1, \mathbf{q}) = \text{dist}([2, 3], [1, 2]) + \text{dist}([4, 5], [3, 4]) = \sqrt{2} + \sqrt{2} = 2\sqrt{2} ]
* **For ( \mathbf{v}\_2 ):**
  + Quantized indices: ( [1, 1] )
  + Asymmetric distance: [ \text{ADC}(\mathbf{v}\_2, \mathbf{q}) = \text{dist}([2, 3], [5, 6]) + \text{dist}([4, 5], [7, 8]) = \sqrt{18} + \sqrt{18} = 2\sqrt{18} ]
* **For ( \mathbf{v}\_3 ):**
  + Quantized indices: ( [2, 2] )
  + Asymmetric distance: [ \text{ADC}(\mathbf{v}\_3, \mathbf{q}) = \text{dist}([2, 3], [9, 10]) + \text{dist}([4, 5], [11, 12]) = \sqrt{98} + \sqrt{98} = 2\sqrt{98} ]
* **For ( \mathbf{v}\_4 ):**
  + Quantized indices: ( [3, 3] )
  + Asymmetric distance: [ \text{ADC}(\mathbf{v}\_4, \mathbf{q}) = \text{dist}([2, 3], [13, 14]) + \text{dist}([4, 5], [15, 16]) = \sqrt{242} + \sqrt{242} = 2\sqrt{242} ]

**Conclusion**

The asymmetric distance computation involves comparing the original query sub-vectors to the quantized sub-vectors of the database, using the centroids from the codebooks. This approach allows for efficient distance computation by leveraging precomputed distances from the codebooks, reducing the computational complexity compared to directly computing full Euclidean distances. The ADC method is particularly useful in scenarios where the dataset is large, and memory efficiency is crucial.

Q. Quicker/Fused ADC optimized post lookuptable computation using AVX512 two vector lookups, lookuptable computation is common b/w ADC, Quick ADC and Quicker/Fused ADC

A. Absolutely, the lookup table computation is a common step across ADC, Quick ADC, and Quicker/Fused ADC. This step involves calculating distances between query sub-vectors and centroids in the codebooks, which are then stored for efficient retrieval during the distance computation process. The key differentiator for Quicker/Fused ADC is the optimization of post-lookup table computation using AVX-512 instructions, which accelerates the process significantly. Let's break down these concepts:

**Common Lookup Table Computation**

1. **Query Processing:**
   * The query vector is split into sub-vectors, similar to the database vectors.
2. **Distance Calculation:**
   * For each query sub-vector, compute the distance to each centroid in the corresponding codebook. This step is performed dynamically for each query.
   * The computed distances are stored in a lookup table, which allows for quick access during the subsequent steps.

**Differences in Post-Lookup Table Computation**

**ADC**

* **Basic Retrieval:**
  + ADC uses the lookup table to retrieve precomputed distances and compute the asymmetric distance between the query vector and each database vector. This involves summing the distances for the corresponding sub-vectors.

**Quick ADC**

* **Optimized Retrieval:**
  + Quick ADC enhances the efficiency of distance retrieval by using optimized data structures, such as hash maps or arrays, to store and access precomputed distances. This reduces computational overhead and speeds up the process.

**Quicker/Fused ADC**

* **Hardware-Accelerated Retrieval:**
  + Quicker/Fused ADC leverages AVX-512 instructions to perform fast lookups and permutations of precomputed distances. This allows for simultaneous processing of multiple data points, significantly speeding up the computation.
  + **Two-Vector Permute:** AVX-512's ability to handle wide vectors enables rapid processing of multiple sub-vectors and centroids in parallel, reducing latency and increasing throughput.

**Conclusion**

While the lookup table computation is a common step across ADC, Quick ADC, and Quicker/Fused ADC, the latter optimizes the post-lookup table computation using AVX-512 instructions. This optimization allows for rapid and efficient processing of distances, making Quicker/Fused ADC particularly suitable for large-scale applications requiring high-speed vector similarity searches. By leveraging hardware capabilities, Quicker/Fused ADC provides a significant performance boost over traditional methods.

Baseline with 500000 queries:-

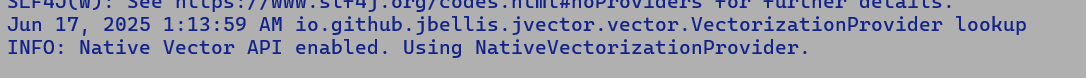
90% time is spent in benchmarking the kernel.

A close-up of a computer screen

AI-generated content may be incorrect.

Quicker ADC performance

***NativeVectorizationProvider***

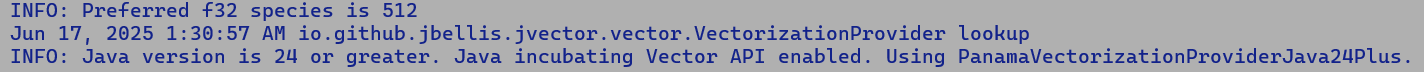
******

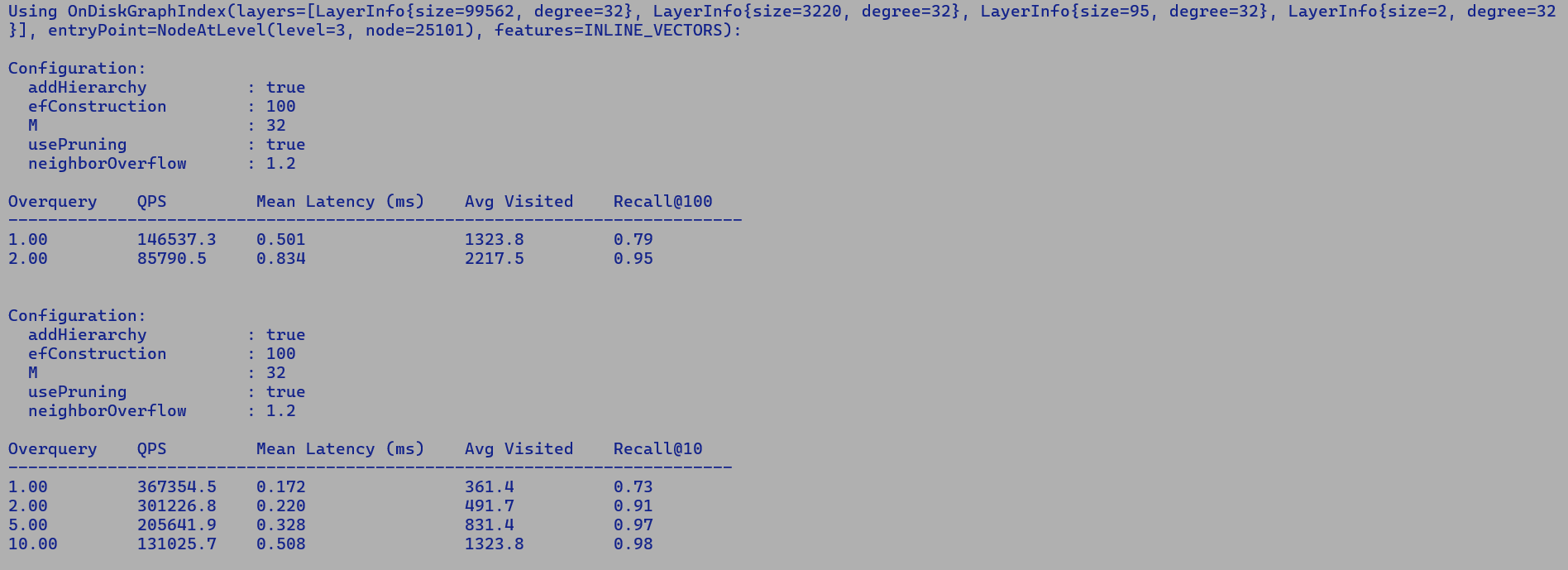
A screenshot of a computer

AI-generated content may be incorrect.



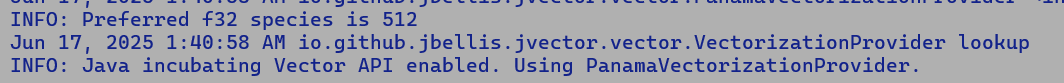
***PanamaVectorizationProviderJava24Plus***

******

******



***PanamaVectorizationProvider***

******

***A screenshot of a computer

AI-generated content may be incorrect.***