Accelerated Approximate Nearest Neighbors Search Through Hierarchical Product Quantization

Ameer M.S. Abdelhadi

Dept. of Electrical & Computer Engineering
University of Toronto

Christos-Savvas Bouganis and George A. Constantinides

Dept. of Electrical & Electronic Engineering
Imperial College London

Nearest Neighbor Search

What's the nearest restaurant to my hotel?

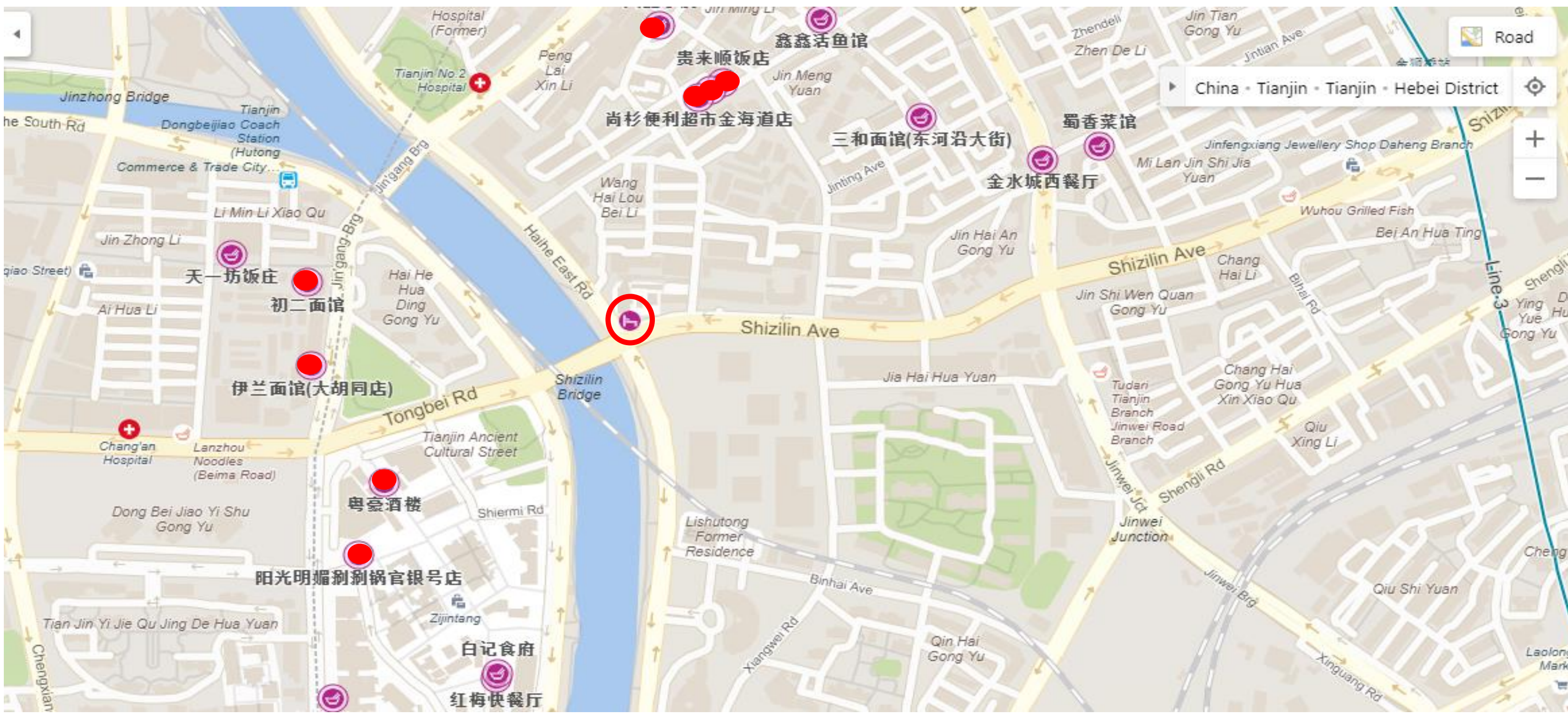
- 2D space, Manhattan distance

Approximate
Nearest Neighbor Search

Find a restaurant whose distance to the hotel is at most $1+\varepsilon$ times the distance from the hotel to its nearest restaurant

k-
Nearest Neighbor Search

Find the k nearest restaurants to the hotel
- e.g. 8 nearest restaurants



k-ANN Applications

- Data compression
- Databases and data mining
- Information retrieval
- Image and video databases
- Machine learning
- Pattern recognition
- Statistics and data analysis
- Recently: Memory-Augmented Neural Networks (Google DeepMind)

Memory-Augmented Neural Networks

- DNNs with local memory (e.g. LSTM)
  - Store the memory via weight adjustment
  - Cannot store data over long time scale
  - Incapable to efficiently solve complex structured tasks
- Solution:
  - Explicit external memory with differentiable attention
  - Allows teaching using end-to-end backpropagation
- Differentiable memory is compute- and memory-intensive, thus is not scalable!

Sparse Memory-Augmented Neural Networks

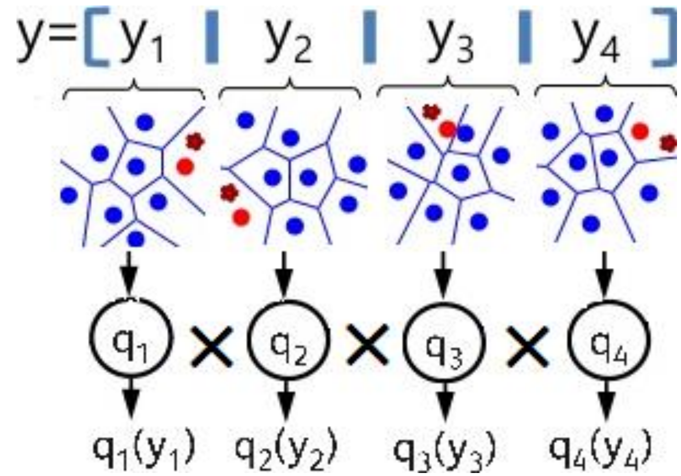
- A DNN with differentiable memory units is not scalable
- The system does a lot of computations just to access a small chunk of memory
- Heavy computational resources are needed to make this scale up
- Scalability in training: unfolding the memory dramatically expands the network
- Solution: sparse reads and writes based on *k*-ANN
  - Online updates are not supported
  - Storage and querying performance are not scalable

Approximate Nearest Neighbors (ANN) Search

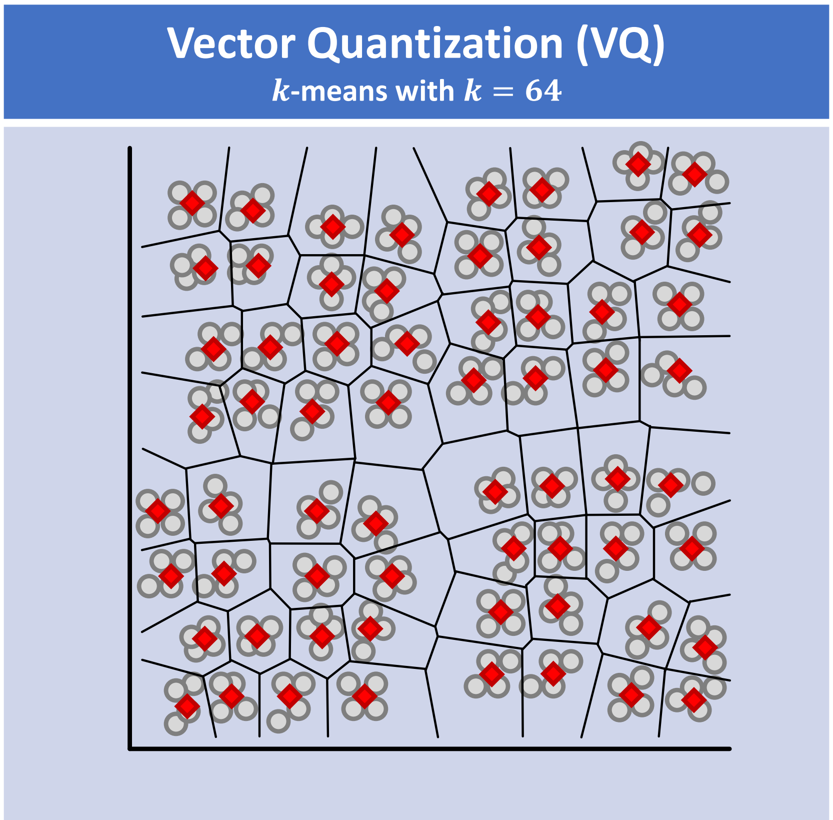
- Leading ANN methods:
  - Space splitting metric trees:
    - branch-and-bound space partitioning, *e.g.* KD-trees, R-trees , K-D-B-tree, VP-trees…
    - Suffer from the "*curse of dimensionality*"!
  - Hashes:
    - Hamming distance is used to approximate similarity
    - e.g. Locality sensitive hashes (LSH)
  - Product quantization (PQ):
    - Quantizes several subspaces separately
- Disadvantages:
  - Software-optimized: hardware acceleration is not supported
  - Memory-intensive: require intensive external memory access
  - Static database: updates require reconstructing the data structure

Product Quantization

- Partitions space into a Cartesian product of low-dimensional subspaces
- Quantizes each partition into clusters
- Data is regenerated from codebooks
- Distance is approximated by summing the distances from each subspace
- Advantages:
    - Dimension-wise scalability
    - High accuracies
    - Amendable to massively-parallel hardware acceleration on FPGAs
        - 1000’s × distributed memories (Block RAM) → distributed codebooks
        - 1000’s × DSPs → FP operations for distance calculation
    - Amendable to support online updates
    - Storage efficient: only quantized data need to be stored

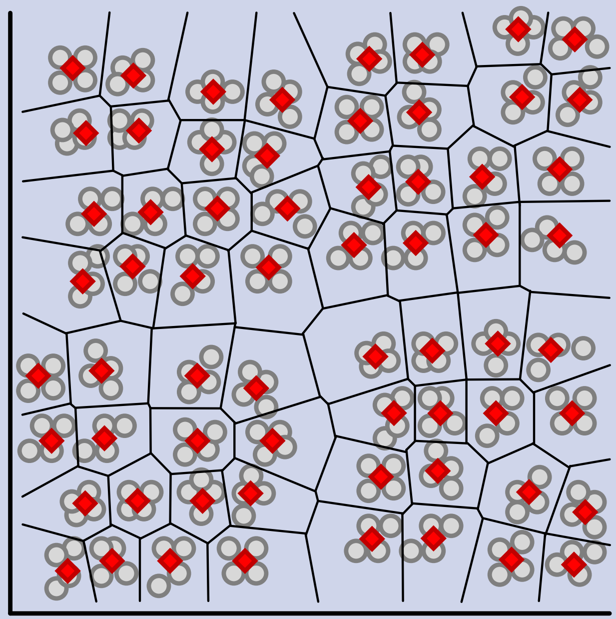
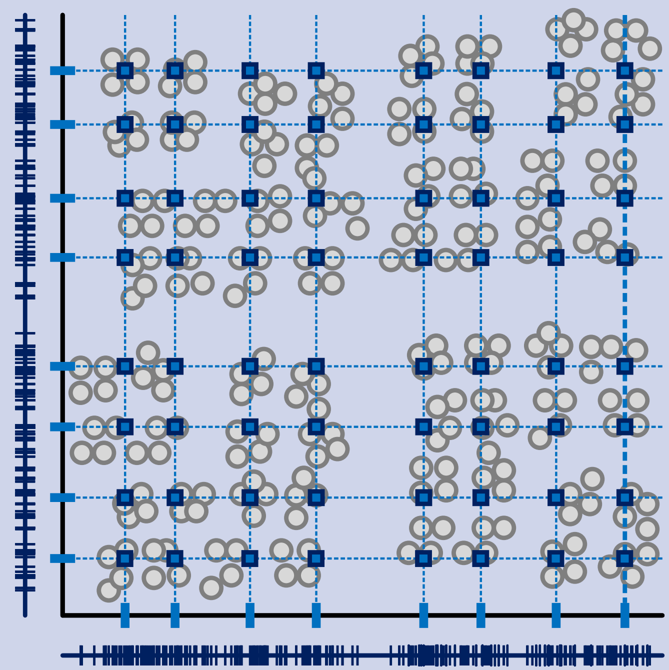


Multi-dimensional quantization:
a two-dimensional example ($D = 2$)



Legend: Data point; VQ centroid;

Multi-dimensional quantization:
a two-dimensional example ($D = 2$)

Vector Quantization (VQ) k-means with $k = 64$	Product Quantization (PQ) k-means for two ($M = 2$) 1D ($\widetilde{D} = 1$) subspaces with $k = 8$
	

Legend: Data point; VQ centroid; PQ centroid; PQ subcode

Hierarchical Product Quantization
Key idea

Objectives:

- Avoid exhaustive search
- Enhance parallelism
- Support online updates

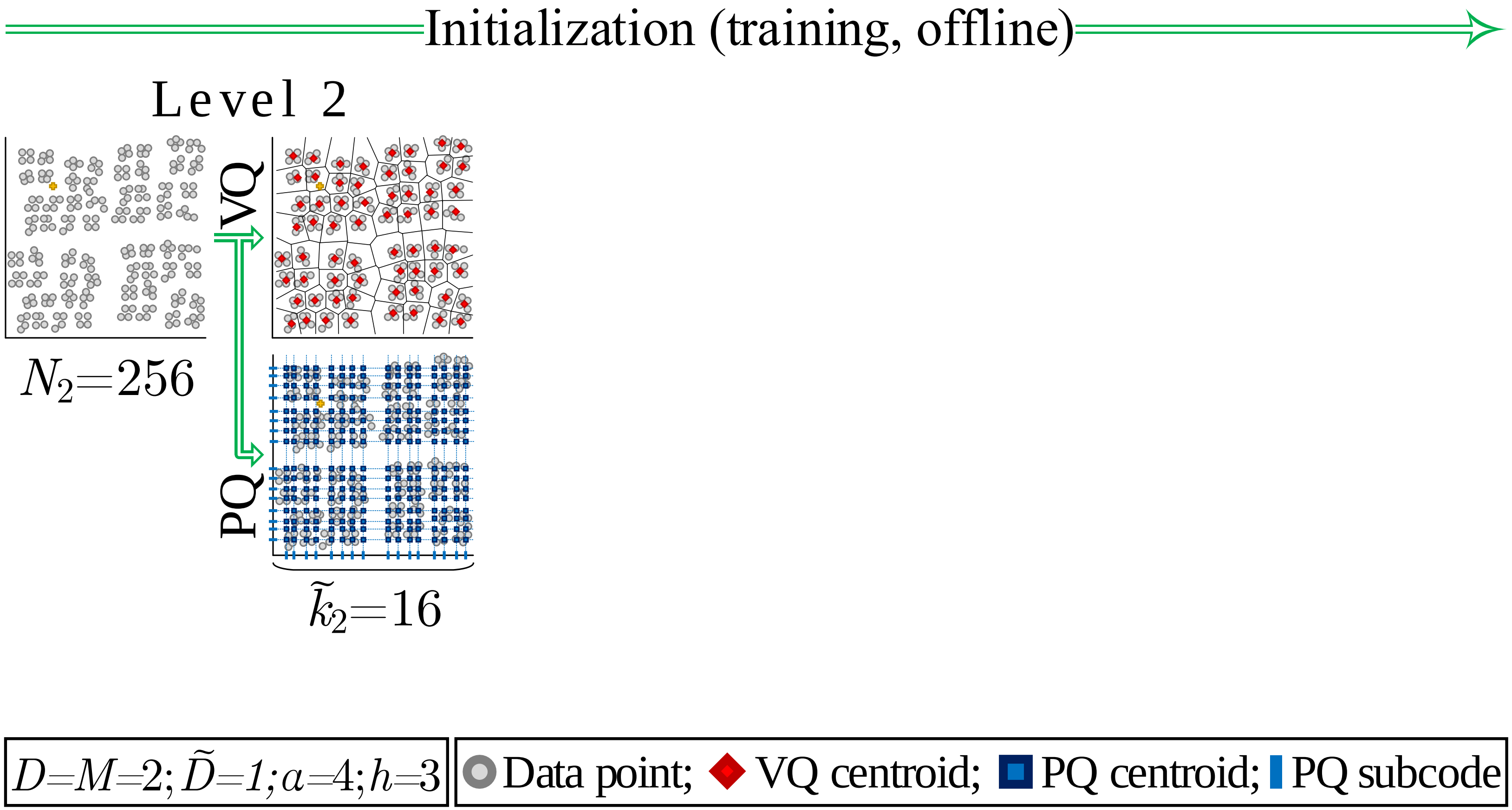
Key idea:

- Enhance PQ by using space partitioning techniques
- Gradually refine search space with hierarchical search
- VQ gradually subdivides the PQ search space

×250 speedup compared to other OpenCL-FPGA & GPU methods

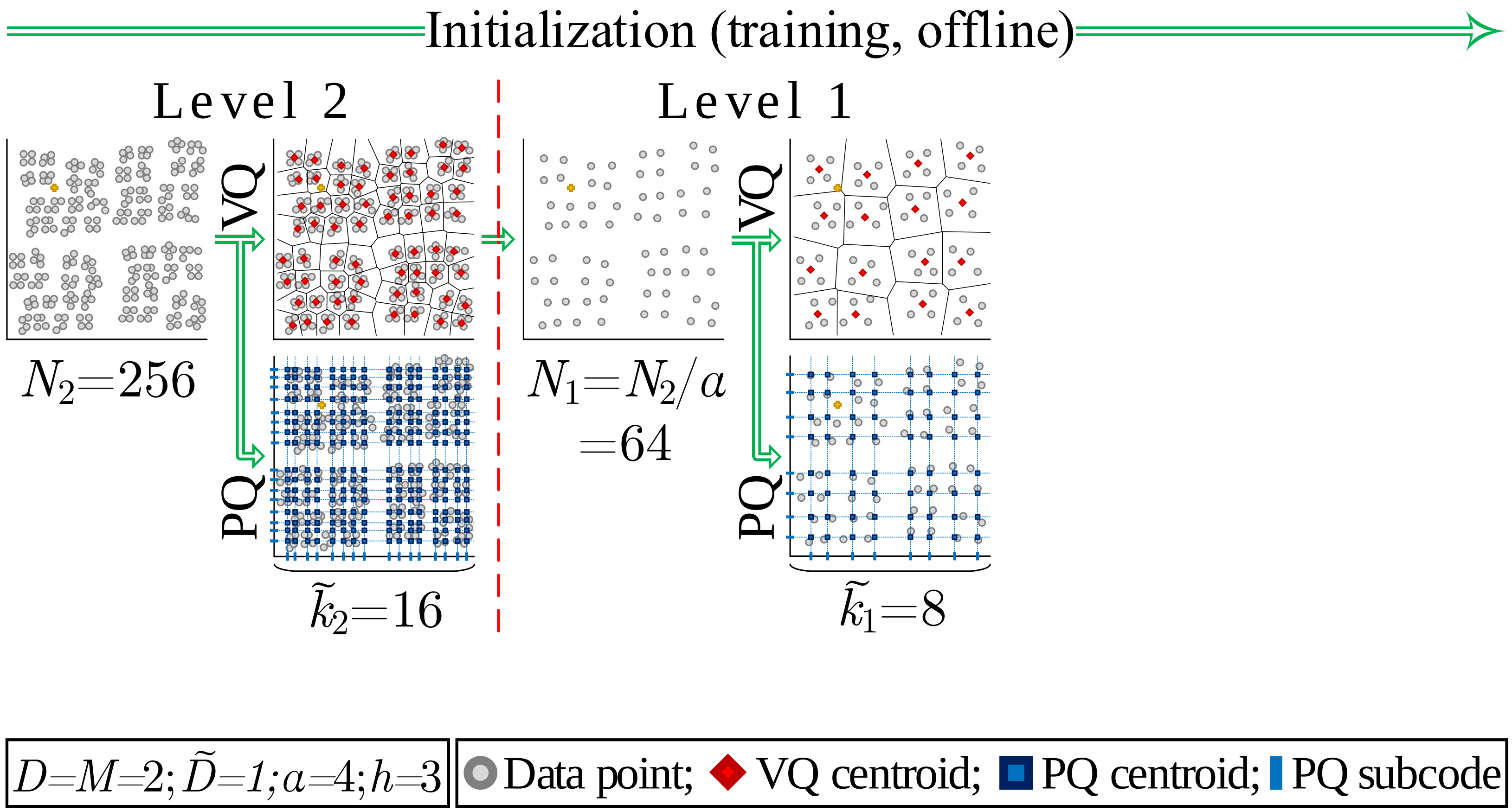
Hierarchical Product Quantization

2D ($D = 2$), 3 Levels ($h = 3$) Example



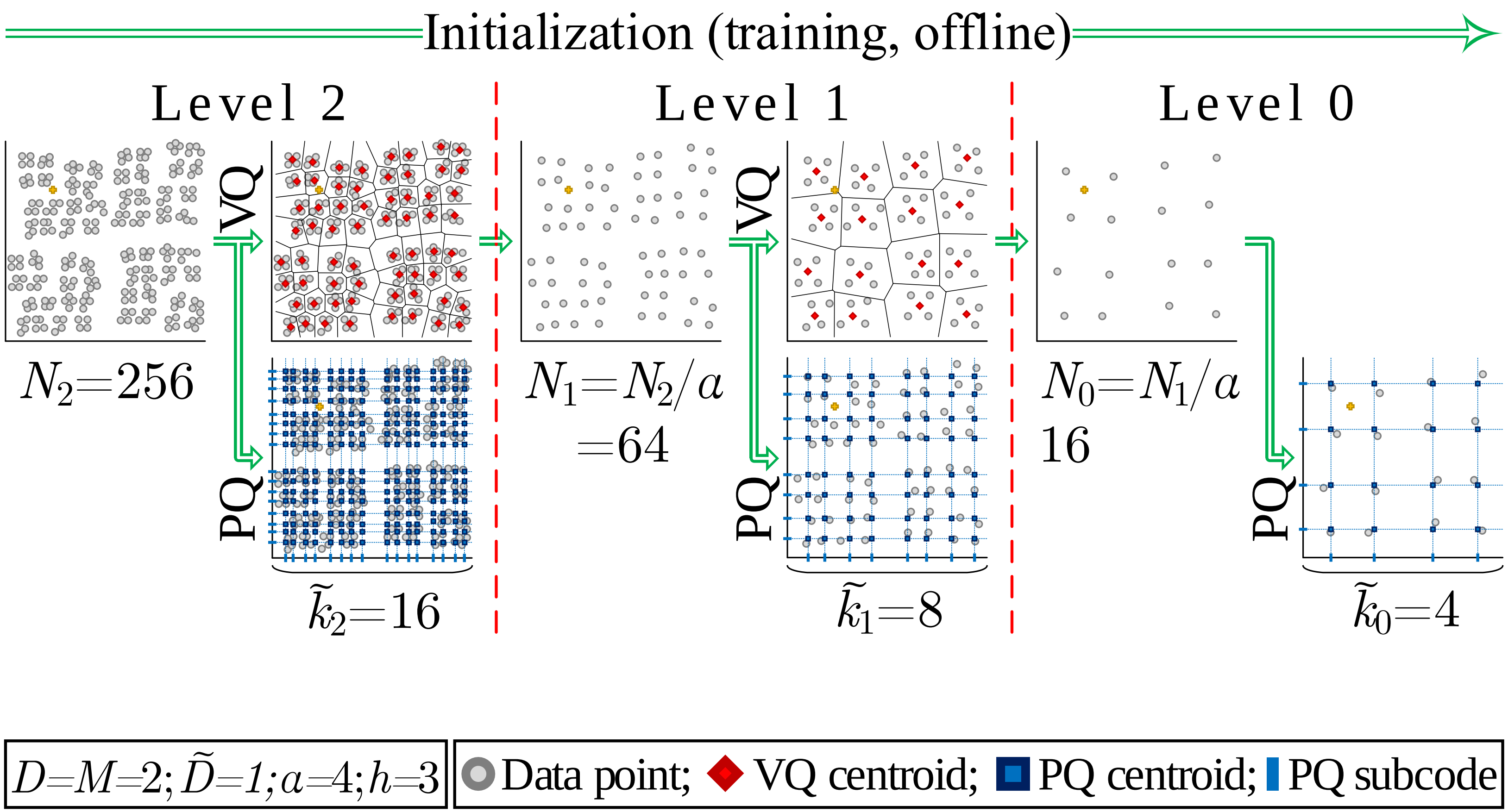
Hierarchical Product Quantization

2D ($D = 2$), 3 Levels ($h = 3$) Example



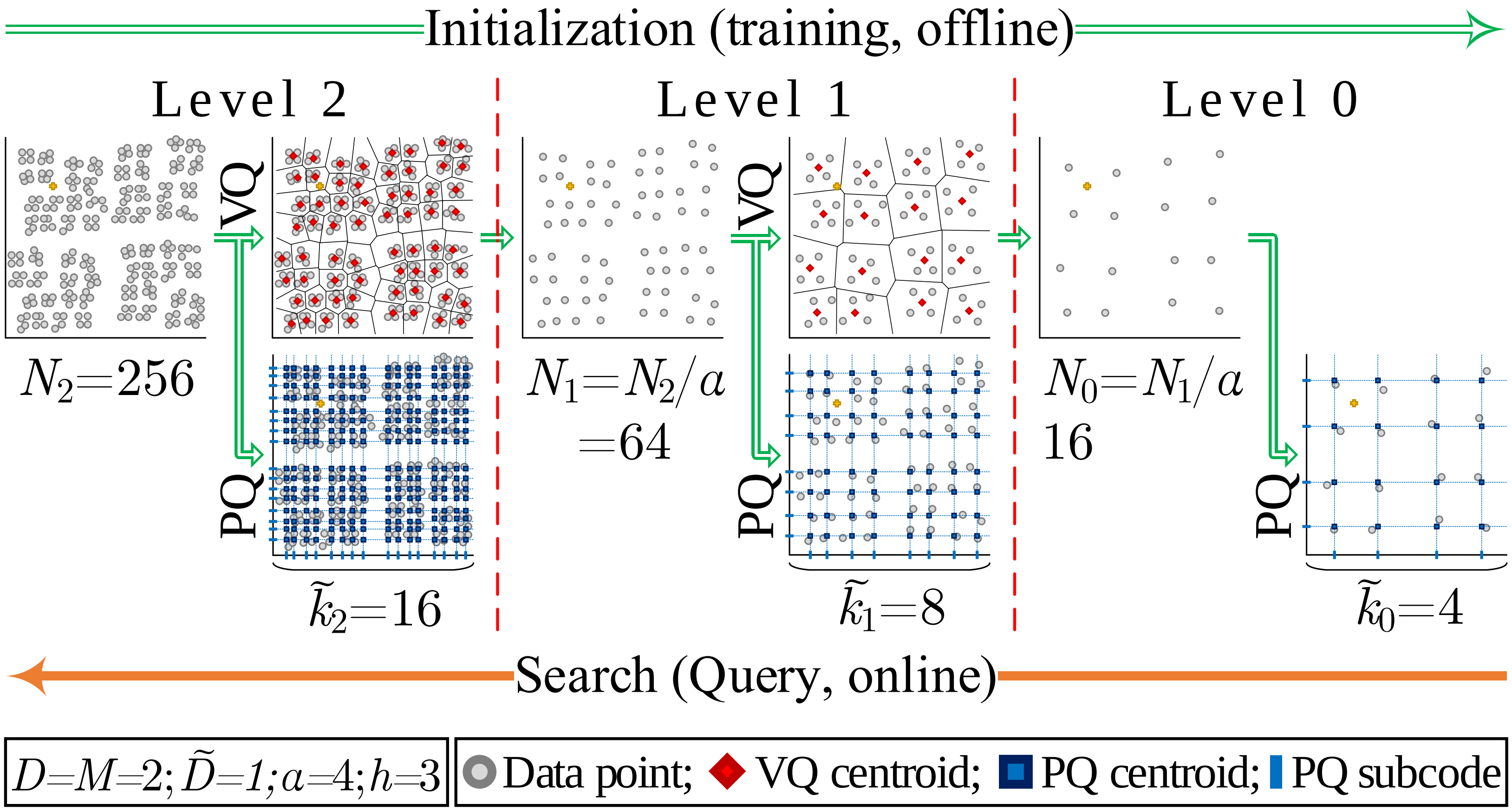
Hierarchical Product Quantization

2D ($D = 2$), 3 Levels ($h = 3$) Example



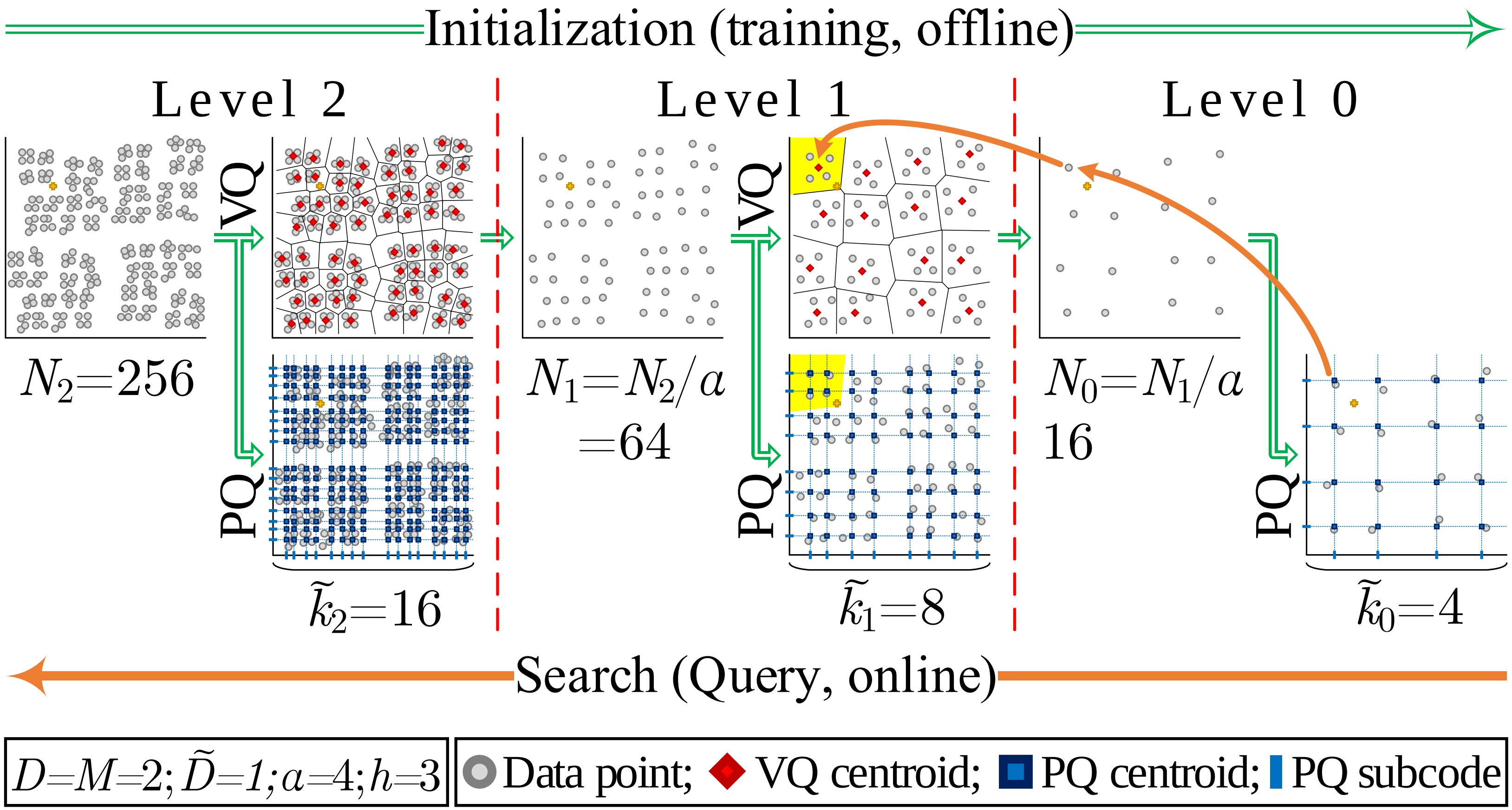
Hierarchical Product Quantization

2D ($D = 2$), 3 Levels ($h = 3$) Example



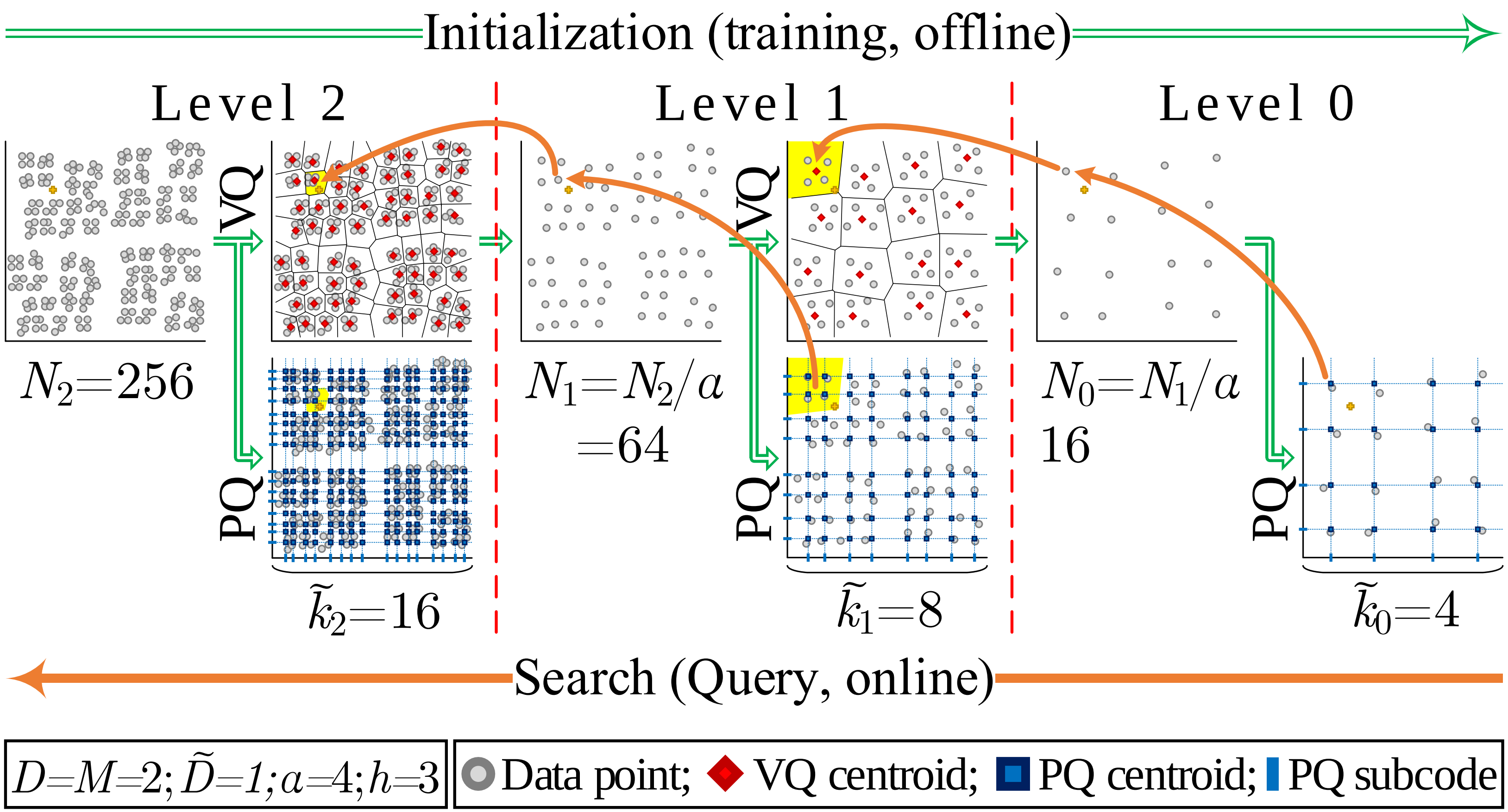
Hierarchical Product Quantization

2D ($D = 2$), 3 Levels ($h = 3$) Example



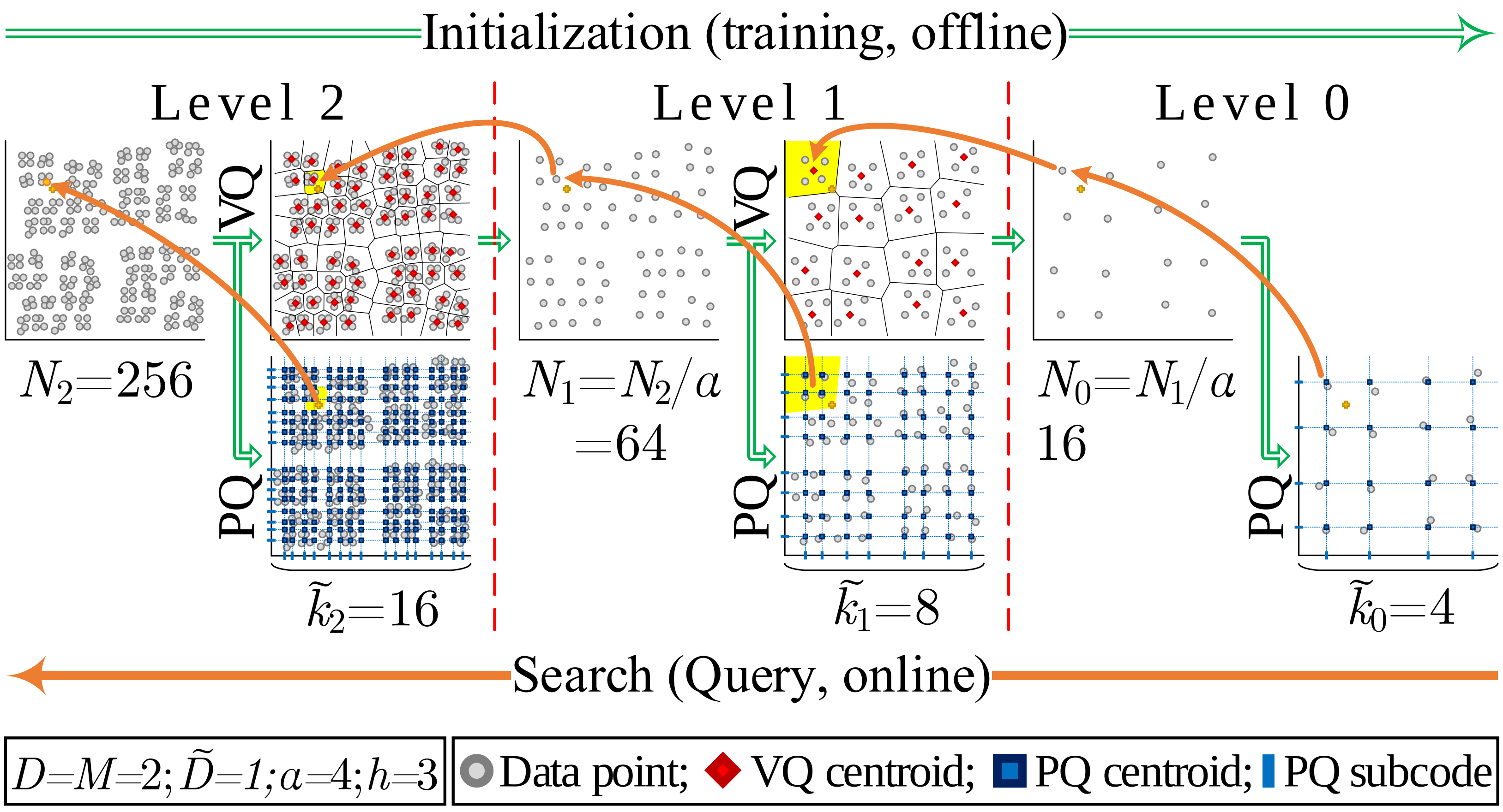
Hierarchical Product Quantization

2D ($D = 2$), 3 Levels ($h = 3$) Example



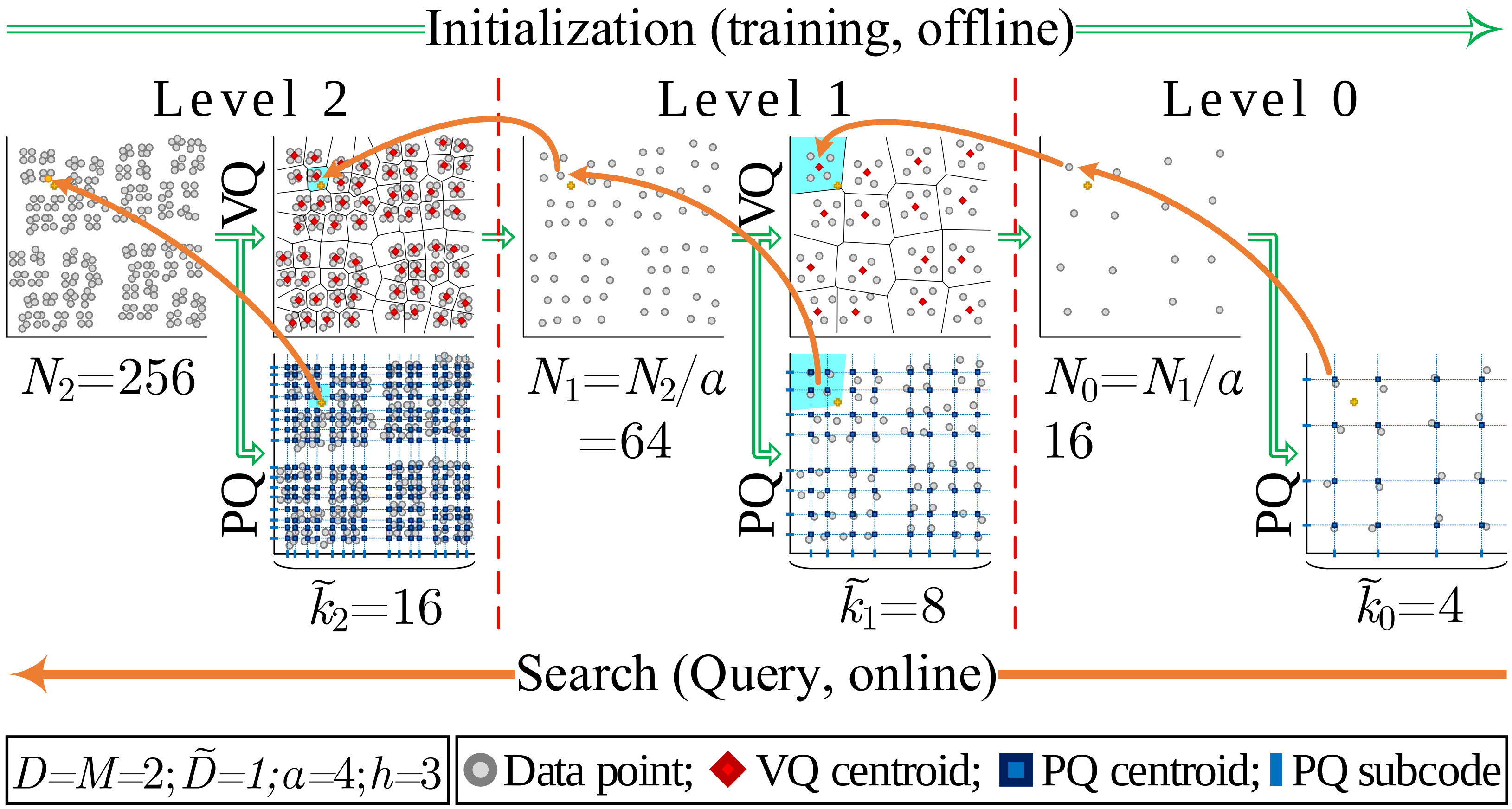
Hierarchical Product Quantization

2D ($D=2$), 3 Levels ($h=3$) Example



Hierarchical Product Quantization

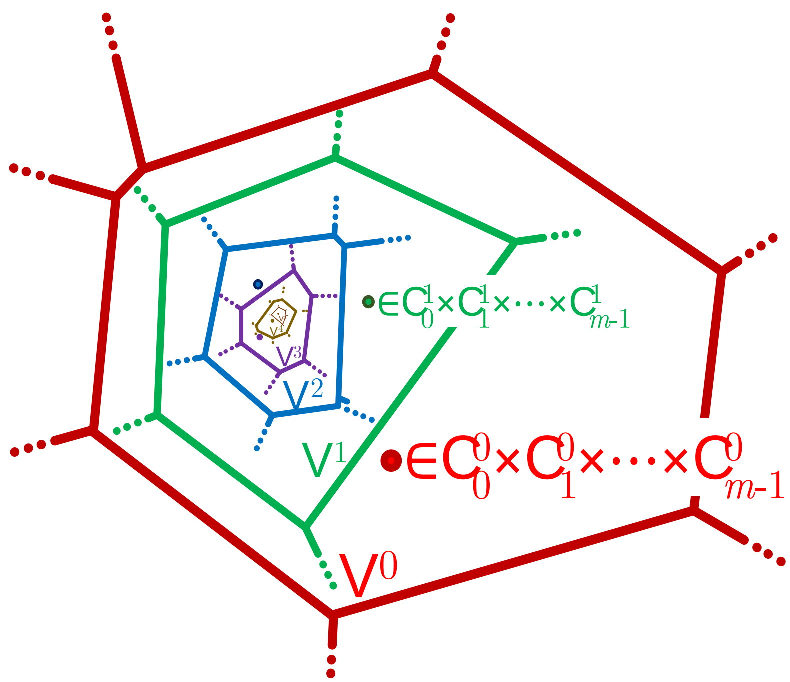
2D ($D = 2$), 3 Levels ($h = 3$) Example



Hierarchical Product Quantization
Querying through Voronoi Cells

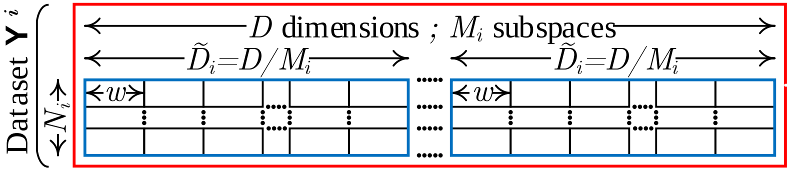
VQ gradually subdivides the PQ search space

PQ is used to search within the refined subspace



The HPQ Data Structure:
A Recursive Definition

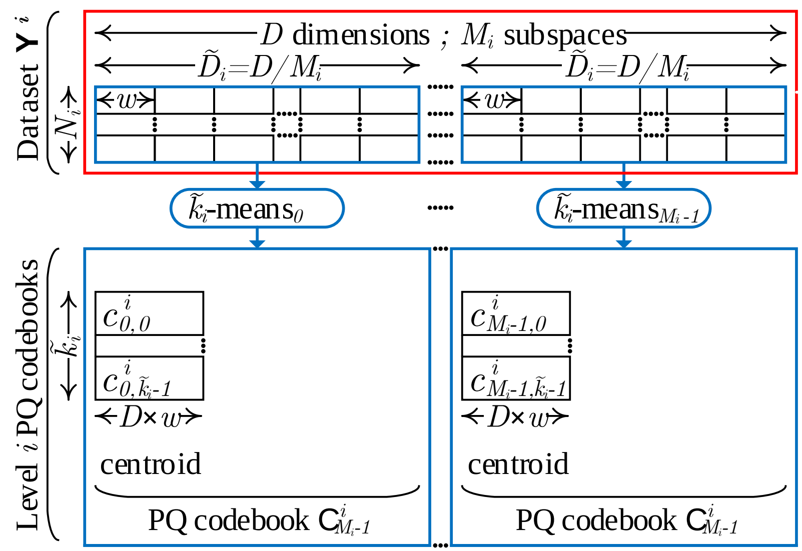
- D-dim dataset



The HPQ Data Structure:
A Recursive Definition

- D-dim dataset

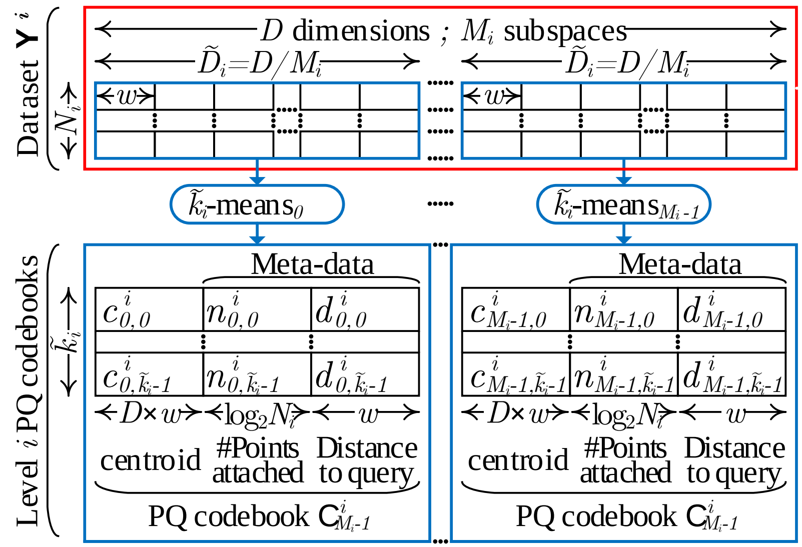
- K-means for each sub-space



The HPQ Data Structure:
A Recursive Definition

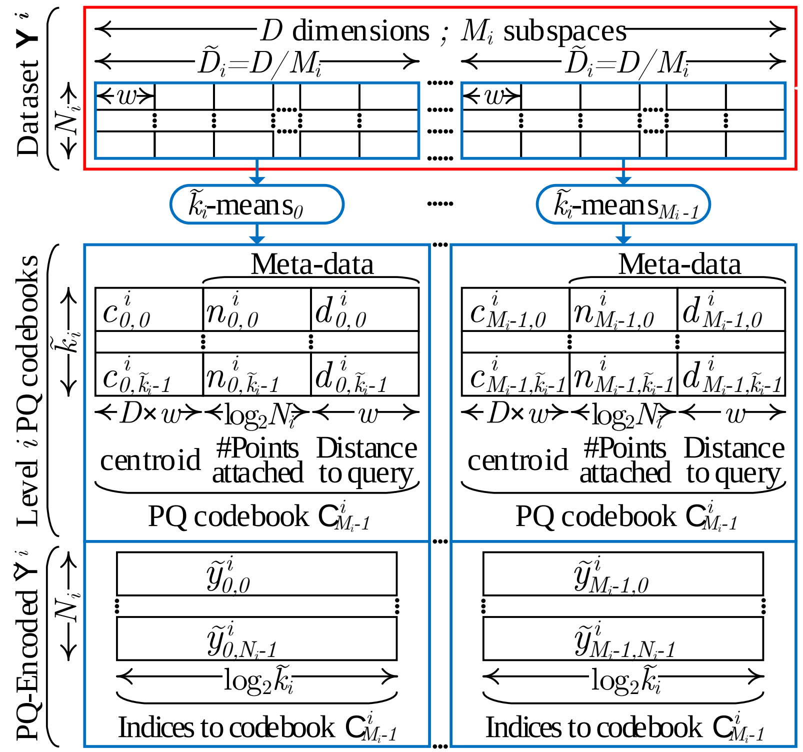
- D-dim dataset

- K-means for each sub-space
- Meta-data: #points & distances



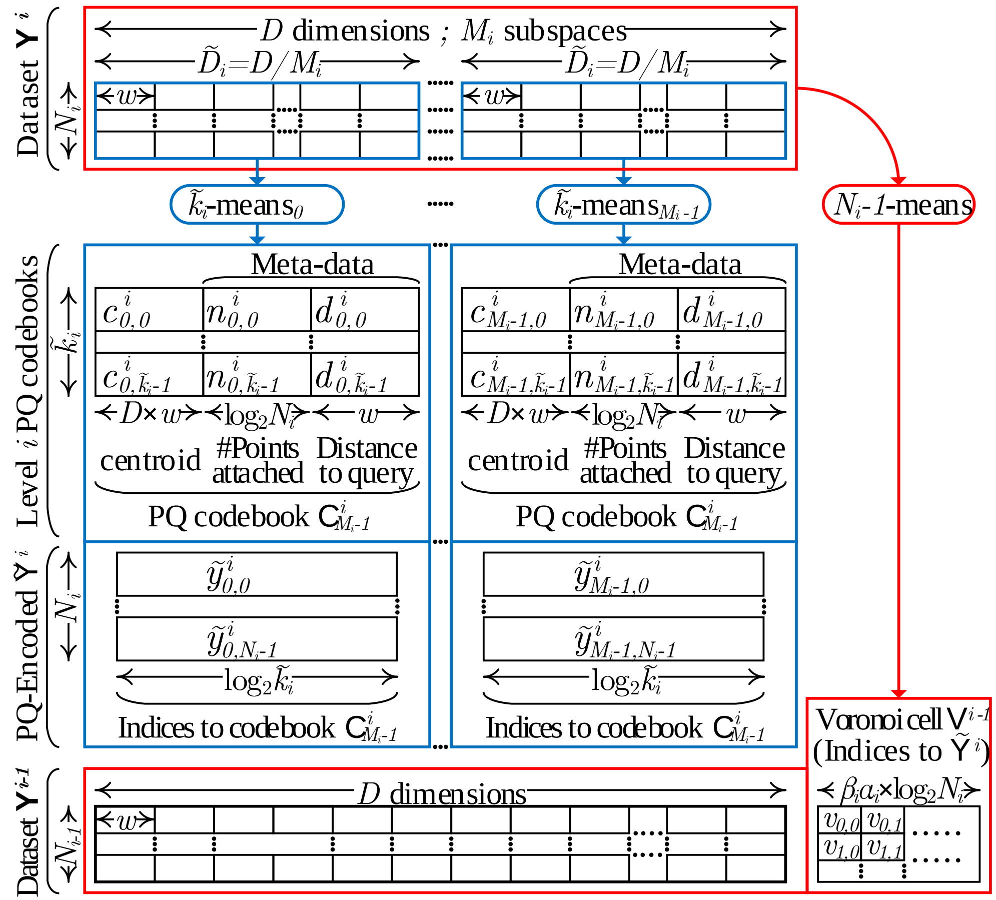
The HPQ Data Structure:
A Recursive Definition

- D-dim dataset
- K-means for each sub-space
- Meta-data: #points & distances
- Encoded dataset



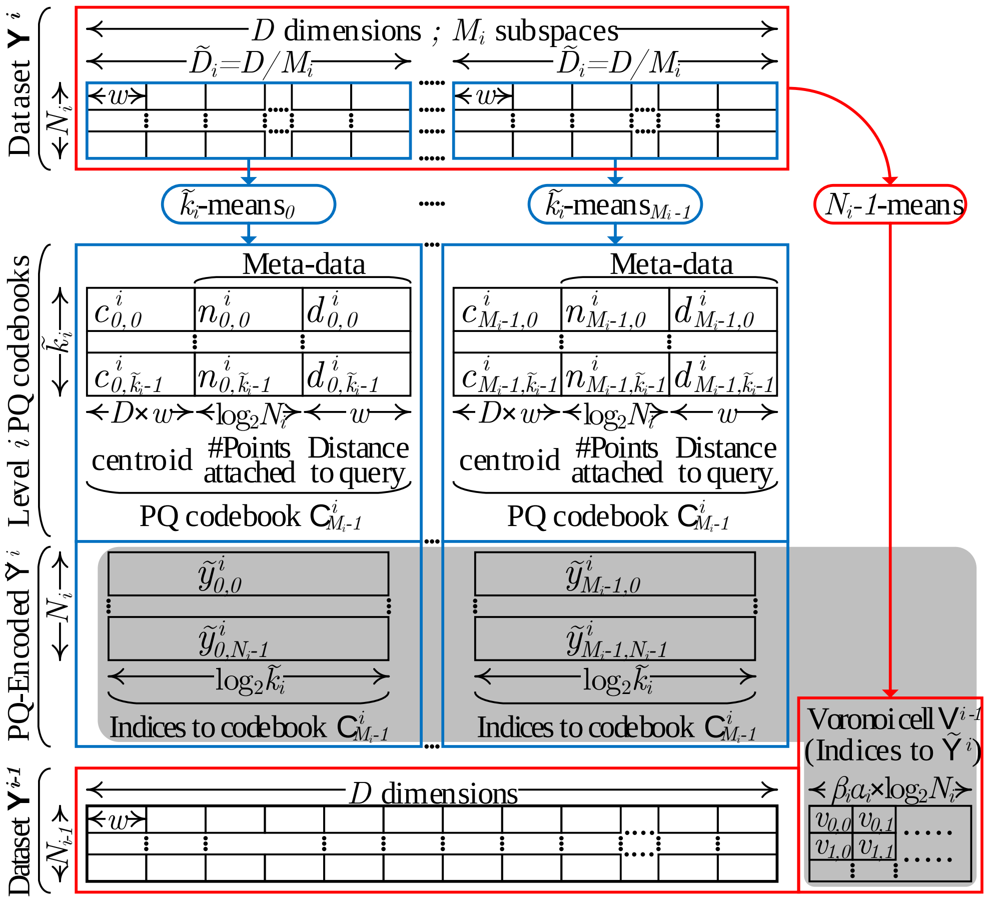
The HPQ Data Structure:
A Recursive Definition

- D-dim dataset
- K-means for each sub-space
- Meta-data: #points & distances
- Encoded dataset
- Vector quantization



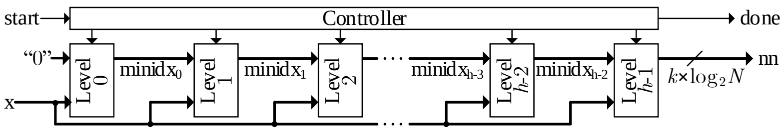
The HPQ Data Structure:
A Recursive Definition

- D-dim dataset
- K-means for each sub-space
- Meta-data: #points & distances
- Encoded dataset
- Vector quantization
- Encoded dataset is stored within Voronoi cells



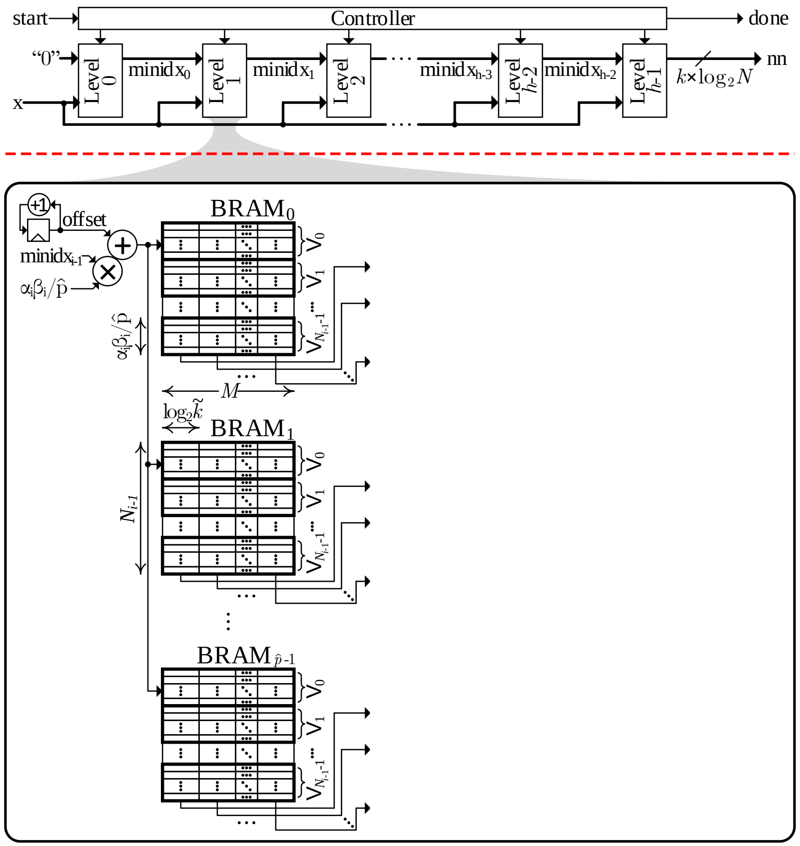
Hardware architecture:

- Layer-wise deep pipeline of refinement indices



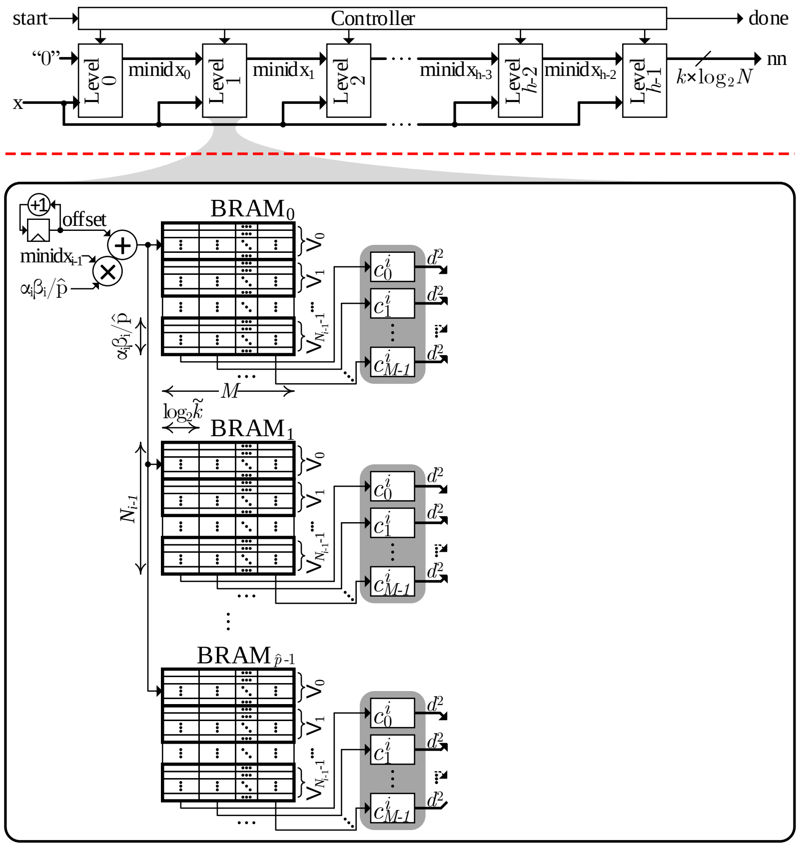
Hardware architecture:

- Layer-wise deep pipeline of refinement indices
- Each Voronoi cell is distributed over multiple BRAMs
- $M \times \tilde{p}$ indices from each BRAM are read in parallel



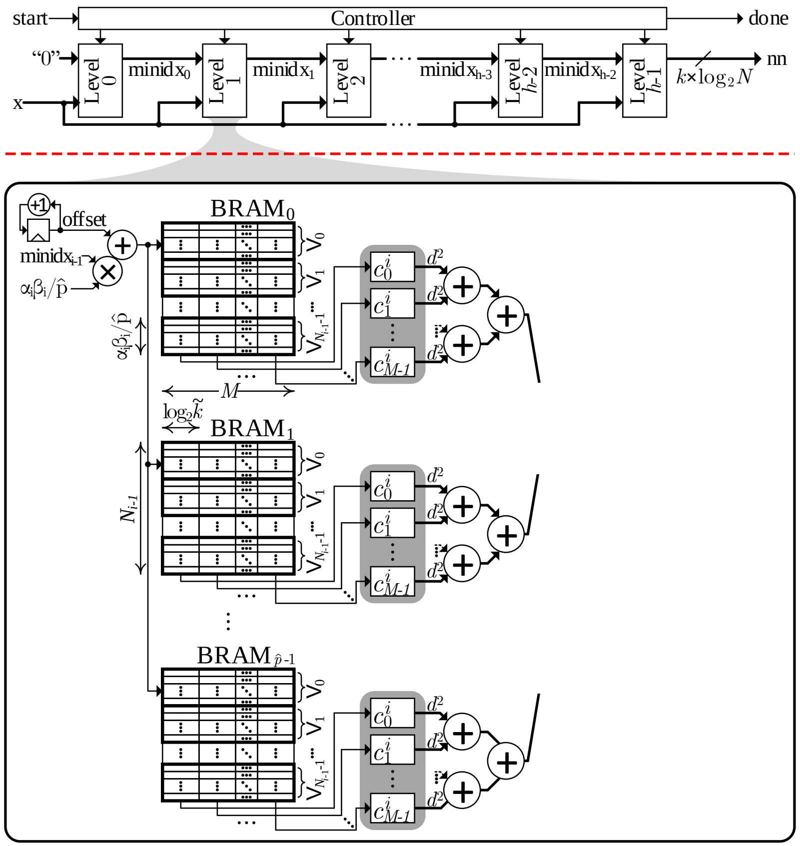
Hardware architecture:

- Layer-wise deep pipeline of refinement indices
- Each Voronoi cell is distributed over multiple BRAMs
- $M \times \tilde{p}$ indices from each BRAM are read in parallel
- Relevant distances of these indices are retrieved from codebooks



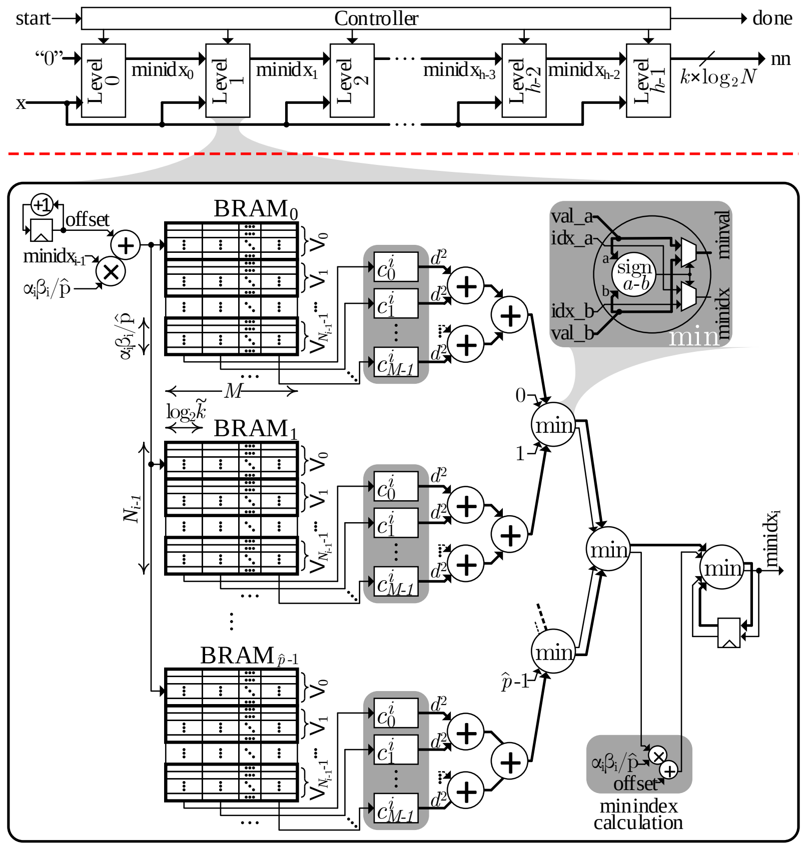
Hardware architecture:

- Layer-wise deep pipeline of refinement indices
- Each Voronoi cell is distributed over multiple BRAMs
- $M \times \tilde{p}$ indices from each BRAM are read in parallel
- Relevant distances of these indices are retrieved from codebooks
- Squared sub-distances are accumulated to find total squared distance



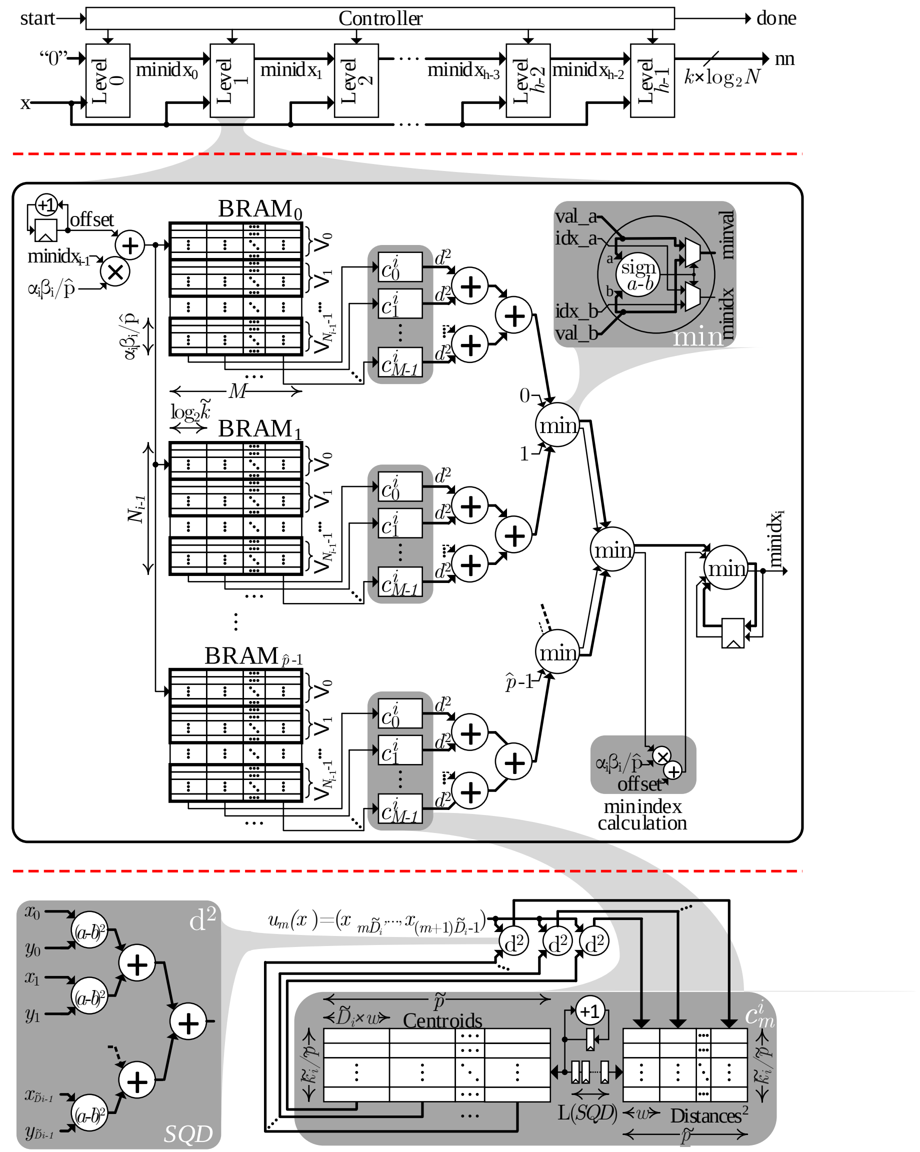
Hardware architecture:

- Layer-wise deep pipeline of refinement indices
- Each Voronoi cell is distributed over multiple BRAMs
- $M \times \tilde{p}$ indices from each BRAM are read in parallel
- Relevant distances of these indices are retrieved from codebooks
- Squared sub-distances are accumulated to find total squared distance
- Distances are compared to find the minimum



Hardware architecture:

- Layer-wise deep pipeline of refinement indices
- Each Voronoi cell is distributed over multiple BRAMs
- $M \times \tilde{p}$ indices from each BRAM are read in parallel
- Relevant distances of these indices are retrieved from codebooks
- Squared sub-distances are accumulated to find total squared distance
- Distances are compared to find the minimum
- Distances in each code book are updated for each query (preprocessing)
- $\hat{p}$ distances are processed in parallel



Experimental Results:
Accuracy Trade-offs

Accuracy			Parallelism				Resources		Performance[a]		
R@100	M	$\tilde{k}$	α	h	$\tilde{p}$	$\hat{p}$	BRAMs Mb	DSPs	F_{max} MHz	Latency us/query	$\frac{1}{\text{Throughput}}$ us/query
0.973	16	64	128	3	8	64	204	5508	334	0.37	0.078
0.89	16	32	128	3	8	64	174	5514	357	0.34	0.062
0.752	8	64	128	3	8	128	110	5703	368	0.33	0.073
0.57	8	32	128	3	8	128	90	5698	412	0.29	0.056

[a] Measured on Stratix10 GX2800 FPGA using on-chip memory only.

- **SIFT1M Benchmark**
  A dataset of 1M 128-dim SIFT vectors
- **Accuracy metric: recalls $R@r$**
  The probability that the nearest neighbor retrieved is ranked within the first rr true nearest neighbors

Experimental Results:
Comparison of Performance and Accuracy

Platform	Method	Latency (us/query)	$\frac{1}{\text{Throughput}}$ (us/query)	R@100
CPU[a]	LOPQ		51.1k	0.97
	IVFPQ		11.2k	0.93
GPU[b]	PQT		20	0.86
	FAISS		20	0.95
OpenCL FPGA[c]	LOPQ		20	0.97
Custom FPGA[d]	HPQ_1	0.85	0.33	0.973
Custom FPGA[e]	HPQ_2	0.37	0.078	0.973

[a] Xeon E5-1630v3 CPU: quad core, 8 threads, 10MB cache, 3.7GHz.
[b] Nvidia GTX Titan Xp GPU: 3840 CUDA cores, 1.6GHz, 12 TFLOPs.
[c] Intel HARPv2: 14 core Broadwell Xeon CPU + Arria10 GX1150 FPGA.
[d] Arria10 GX1150 FPGA: 427K ALMs, 53Mb BRAMs, and 1518 DSPs. Optimal design parameters: $(M, \tilde{k}, \alpha, h, \tilde{p}, \hat{p}) = (16, 64, 16, 5, 1, 4)$
[e] Stratix10 GX2800 FPGA: 933K ALMs, 229Mb BRAMs, and 5760 DSPs. Optimal design parameters: $(M, \tilde{k}, \alpha, h, \tilde{p}, \hat{p}) = (16, 64, 128, 3, 8, 64)$

×250 speedup compared to other OpenCL-FPGA & GPU methods

Future Work

- Hierarchical Product Quantization with online updates
- Integration with memory-augmented neural networks

Thank You!