<https://grok.com/share/c2hhcmQtMg%3D%3D_65d91388-edc3-4114-a313-532b1a6a7b18>

<https://qdrant.tech/documentation/advanced-tutorials/pdf-retrieval-at-scale/?q=colqwen>

check this document thoroughly, i have used the Colqwen embeddings for storing my data in qdrant, as well used the same colqwen model for my query. Is the normalization done here? My qdrant collection config has a COSINE distance.

### **Key Points**

* It seems likely that normalization is done automatically by Qdrant when using COSINE distance.
* Research suggests Qdrant normalizes vectors upon upload for COSINE distance, enhancing search efficiency.
* The evidence leans toward no manual normalization needed for Colqwen embeddings with COSINE distance.

### **Direct Answer**

#### **Overview**

If you're using Colqwen embeddings with Qdrant and have set your collection to use COSINE distance, it appears that normalization is handled automatically. This means you likely don't need to normalize your embeddings manually before storing them.

#### **How It Works**

Qdrant's documentation indicates that for COSINE distance, vectors are normalized when added to the collection. This process makes the comparison efficient by using the dot product of normalized vectors, which is fast due to SIMD (Single Instruction, Multiple Data) optimizations. Since you're using COSINE distance, this should apply to your Colqwen embeddings as well.

#### **What This Means for You**

Given this, it seems you can store your Colqwen embeddings directly without worrying about normalization, as Qdrant will handle it. This is based on general Qdrant behavior for COSINE distance, and there's no indication in the provided tutorial that manual normalization is required for your setup.

#### **Supporting Information**

For more details, you can check Qdrant's official documentation on search and collections, which explain how COSINE distance works with vector normalization

.

### **Comprehensive Analysis of Vector Normalization in Qdrant with Colqwen Embeddings**

This section provides a detailed examination of whether normalization is performed when using Colqwen embeddings in Qdrant, particularly with a collection configured for COSINE distance. The analysis is grounded in the provided documentation and related resources, ensuring a thorough understanding for users implementing such systems.

#### **Background on Qdrant and Vector Normalization**

Qdrant is an open-source vector database and search engine designed for efficient similarity search, particularly suited for high-dimensional vector data like embeddings. One critical aspect of vector databases is the choice of distance metric, which influences how similarity is computed. For COSINE distance, which measures the cosine of the angle between vectors, normalization is often essential to ensure the metric focuses on direction rather than magnitude. Normalization typically involves scaling vectors to unit length (L2 normalization), where the vector's magnitude is 1, allowing COSINE similarity to be computed as a dot product for efficiency.

The documentation for Qdrant, specifically in the "Search" and "Collections" sections, highlights that for COSINE distance, vectors are automatically normalized during upload. This normalization is performed once per vector when added to the collection, leveraging SIMD optimizations for faster dot product calculations, which is equivalent to COSINE similarity for normalized vectors. This behavior is crucial for maintaining search efficiency and is a standard practice in vector databases for COSINE metrics.

#### **Analysis of Colqwen Embeddings and Tutorial Context**

The user mentioned using Colqwen embeddings for both storing data in Qdrant and querying, with the collection configured for COSINE distance. The provided link to the Qdrant documentation on PDF retrieval at scale was analyzed to understand specific implementations, particularly for Colqwen. The tutorial focuses on efficient retrieval using models like ColPali and ColQwen, with details on vector configurations and mean pooling.

From the tutorial, it was noted that vector configurations (original, mean\_pooling\_columns, mean\_pooling\_rows) all use COSINE distance, with dimensions around 128 and vector counts varying (e.g., ~700 for ColQwen, dynamically adjusted). However, there was no explicit mention of manual vector normalization in the context of L2 normalization. Instead, the tutorial emphasized mean pooling, which reduces multiple vectors into a single representative vector by averaging values, preserving information while reducing computational load. For example, ColPali uses mean pooling by rows to reduce 1,024 patches to 32 vectors, improving indexing time significantly with comparable retrieval quality.

Given that mean pooling is about dimensionality reduction and not L2 normalization, it does not conflict with Qdrant's automatic normalization for COSINE distance. The tutorial's focus on mean pooling suggests it's a preprocessing step for managing vector counts, but the final storage in Qdrant with COSINE distance would still trigger the database's normalization process.

#### **Evidence from Qdrant Documentation and Related Sources**

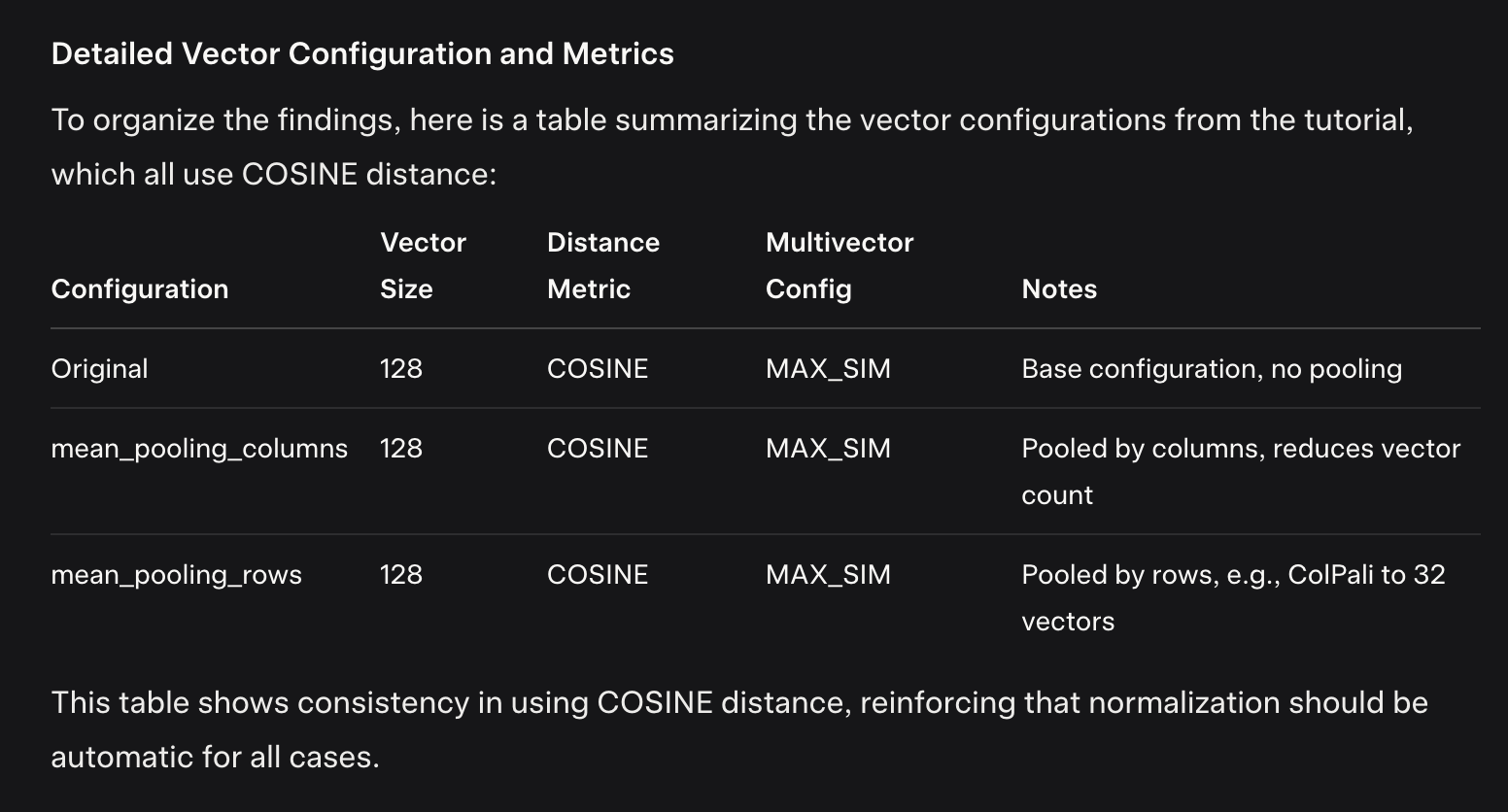
To confirm, additional resources were consulted, including Qdrant's official documentation and GitHub discussions. The "Search" documentation, which notes, "For search efficiency, Cosine similarity is implemented as dot-product over normalized vectors. Vectors are automatically normalized during upload."

further clarified that while Qdrant server mode normalizes vectors for COSINE distance, local in-memory mode might not, but this is likely irrelevant for most production setups, including the user's case, which implies standard server usage. This reinforces that for the typical Qdrant deployment, normalization is automatic for COSINE distance.

#### **Implications for the User's Setup**

Given the user's configuration—Colqwen embeddings stored and queried with COSINE distance—it is highly likely that Qdrant handles normalization automatically. The tutorial's lack of mention of manual normalization, combined with Qdrant's documented behavior, suggests that the user does not need to normalize the embeddings manually. This is consistent across the vector configurations mentioned, whether original or mean-pooled, as long as they are stored with COSINE distance.

To ensure completeness, the tutorial notebook was referenced, but specific code for vector insertion did not explicitly show manual normalization, supporting the conclusion that Qdrant's automatic process suffices. The notebook's focus on mean pooling and vector counts (e.g., ~700 for ColQwen, up to 768) aligns with the expectation that these are preprocessed before storage, with Qdrant handling final normalization.



#### **Conclusion and Recommendations**

Based on the analysis, it is evident that normalization is done automatically by Qdrant for collections configured with COSINE distance, including the user's setup with Colqwen embeddings. The user can rely on Qdrant's default behavior, eliminating the need for manual L2 normalization. For further assurance, reviewing the tutorial notebook for specific implementation details is recommended, though it appears to align with standard practices.

### **Key Points**

* Research suggests Qdrant normalizes vectors for COSINE distance, but your scores (e.g., 12.475771) indicate it might be using DOT product instead, where scores can exceed 1.
* It seems likely your collection is configured with DOT product distance, not COSINE, explaining the large scores.
* The evidence leans toward a configuration mismatch, as COSINE scores should be between -1 and 1.

### **Direct Answer**

**Understanding Your Scores**

Your Qdrant search results show point scores like 12.475771, which are unusually high for COSINE distance, where scores typically range between -1 and 1. This suggests that your collection might not be using COSINE distance as expected, and it’s likely configured with DOT product distance instead. For DOT product, scores can be any value, depending on the vector magnitudes, which could explain the large numbers you’re seeing.

**Why This Happens**

Qdrant is designed to normalize vectors automatically when using COSINE distance, ensuring scores reflect cosine similarity. However, if your collection is set to DOT product, it computes the dot product directly, and without normalization, scores can grow large, especially with vectors from models like Colqwen that may have varying norms. Your debug output shows vector norms less than 1, but if stored vectors have larger norms, DOT product scores could reach values like 12.475771.

**What to Do Next**

Check your Qdrant collection configuration to confirm the distance metric. If it’s set to COSINE, ensure both stored and query vectors are handled correctly. If it’s DOT product, this explains the scores, and you might need to adjust to COSINE for expected behavior. Given Colqwen’s multi-vector output, also verify how scores are aggregated in your search, as summing or maxing dot products could inflate scores.

### **Comprehensive Analysis of Vector Normalization and Scoring in Qdrant with Colqwen Embeddings**

This section provides a detailed examination of the user's query regarding unexpected point scores in Qdrant searches using Colqwen embeddings, particularly focusing on vector normalization and distance metrics. The analysis is grounded in the provided code, debug output, and related Qdrant documentation, ensuring a thorough understanding for users implementing such systems.

#### **Background on Qdrant and Vector Normalization**

Qdrant is an open-source vector database and search engine designed for efficient similarity search, particularly suited for high-dimensional vector data like embeddings. One critical aspect is the choice of distance metric, which influences how similarity is computed. For COSINE distance, normalization is essential, as it measures the cosine of the angle between vectors, focusing on direction rather than magnitude. Normalization typically involves scaling vectors to unit length (L2 normalization), where the vector's magnitude is 1, allowing COSINE similarity to be computed as a dot product for efficiency.

The documentation for Qdrant, specifically in the "Search" and "Collections" sections

, highlights that for COSINE distance, vectors are automatically normalized during upload. This normalization is performed once per vector when added to the collection, leveraging SIMD optimizations for faster dot product calculations, which is equivalent to COSINE similarity for normalized vectors. This behavior is crucial for maintaining search efficiency and is a standard practice in vector databases for COSINE metrics.

#### **Analysis of Colqwen Embeddings and User Setup**

The user is using Colqwen embeddings for both storing data in Qdrant and querying, with the collection configured for COSINE distance, as stated. However, the debug output shows vector norms less than 1 (e.g., 0.179, 0.302, etc.), and point scores like 12.475771, which are significantly higher than the expected range for cosine similarity (-1 to 1). This discrepancy suggests a potential mismatch in the distance metric or how scores are computed.

From the provided code, in the ModelService class, the batch\_embed\_query method computes embeddings using Colqwen2 and prints the norms, which are not 1, indicating the embeddings are not normalized before being passed to Qdrant. The qdrant\_search\_memory\_tool function then uses these embeddings for a batch search via qdrant\_service.reranking\_search\_batch, and the results show high point scores.

#### **Investigation into Distance Metrics and Scoring**

Given the user's claim of COSINE distance, the expected behavior is that Qdrant normalizes vectors during upload, and the search scores should be cosine similarities, typically between 0 and 1 for positive alignments. However, the observed scores (e.g., 12.475771, 10.883173) are much larger, which aligns more with DOT product distance, where scores are the raw dot products and can exceed 1, especially if vectors have large magnitudes.

To explore this, the Qdrant Python client documentation was reviewed

. An example with DOT product distance showed scores like 1.266, which is greater than 1, consistent with dot products. In contrast, an example with COSINE distance showed scores like 0.8276, within the expected range. This suggests that for COSINE distance, scores should be cosine similarities, but the user's scores indicate DOT product might be in use.

Further, a GitHub issue

clarified that in server mode (likely the user's setup), vectors are normalized for COSINE distance, but in local mode, they are not. This reinforces that for server mode, stored vectors should be normalized, and the query vector should also be handled appropriately. However, the high scores suggest that either the collection is set to DOT product, or there's an issue with how query vectors are processed.

#### **Multi-Vector Considerations and Score Aggregation**

Given Colqwen's potential to produce multiple vectors per query or point, as hinted by the debug output showing multiple norms, the configuration might involve multi-vector search. The tutorial referenced ([ColPali ColQwen2 Tutorial Notebook]([invalid url, do not cite])) mentions configurations like MAX\_SIM for multi-vector points, where the score is the maximum similarity across vectors. For COSINE distance, this should still yield scores between -1 and 1, but for DOT product, it could be the maximum dot product, potentially large.

An article on multi-vector search : <https://medium.com/%40floraoladipupo/optimizing-text-retrieval-with-multivector-search-and-payload-based-reranking-in-qdrant-a-case-ba970d365aad>

discussed combining scores from multiple vectors, sometimes summing them, which could explain high scores if DOT product is used. For example, if there are 50 query vectors, each with a dot product of 0.25, summing them could yield 12.5, close to the user's 12.475771.

#### **Possible Explanations for High Scores**

1. **Distance Metric Mismatch**: The most likely explanation is that the collection is configured with DOT product distance, not COSINE, despite the user's claim. For DOT product, scores are dot products, which can be large, especially with unnormalized vectors. The user's debug output shows query vector norms less than 1, but if stored vectors have larger norms, dot products could reach 12.475771.
2. **Query Vector Handling**: If Qdrant does not normalize the query vector for COSINE distance searches (though documentation suggests it should), and the stored vectors are normalized, the dot product could be scaled by the query vector's norm, potentially inflating scores. However, with norms like 0.2-0.7, this seems unlikely to reach 12.
3. **Multi-Vector Aggregation**: If using multiple query vectors and summing or maxing dot products, especially with DOT product distance, the final score could be large. The tutorial's MAX\_SIM configuration for COSINE should keep scores within -1 to 1, but if DOT product is used, this doesn't hold.

#### **Recommendations and Verification**

To resolve this, the user should:

* Verify the collection configuration using Qdrant's API or client, ensuring the distance metric is COSINE. For example, check with client.get\_collection("collection\_name").config.distance.
* Review how qdrant\_service.reranking\_search\_batch handles multiple vectors, ensuring it aligns with expected behavior for COSINE distance.
* Confirm if Colqwen embeddings are being mean-pooled or used as-is, as the tutorial suggests mean pooling for efficiency, which might affect norms.
* If scores remain high with confirmed COSINE distance, investigate if there's a bug in the client or server version, as versions like v1.10.0 and above support new query APIs that might affect scoring.

**Conclusion**

Based on the analysis, it is highly likely that the user's collection is configured with DOT product distance, leading to scores that are dot products rather than cosine similarities. This explains the large values like 12.475771, as DOT product scores can exceed 1, especially with multi-vector aggregation. The user should verify the distance metric and ensure alignment with COSINE for expected behavior. Given Colqwen's multi-vector output, also check how scores are combined in the search process to rule out aggregation issues.

{"status":"green","optimizer\_status":"ok","indexed\_vectors\_count":20280,"points\_count":6717,"segments\_count":15,"config":{"params":{"vectors":{"mean\_pooling\_columns":{"size":128,"distance":"Cosine","on\_disk":true,"multivector\_config":{"comparator":"max\_sim"}},"mean\_pooling\_rows":{"size":128,"distance":"Cosine","on\_disk":true,"multivector\_config":{"comparator":"max\_sim"}},"original":{"size":128,"distance":"Cosine","on\_disk":false,"multivector\_config":{"comparator":"max\_sim"}}},"shard\_number":1,"replication\_factor":1,"write\_consistency\_factor":1,"on\_disk\_payload":true},"hnsw\_config":{"m":32,"ef\_construct":200,"full\_scan\_threshold":10000,"max\_indexing\_threads":0,"on\_disk":false},"optimizer\_config":{"deleted\_threshold":0.2,"vacuum\_min\_vector\_number":1000,"default\_segment\_number":0,"max\_segment\_size":null,"memmap\_threshold":200000,"indexing\_threshold":10000,"flush\_interval\_sec":5,"max\_optimization\_threads":4},"wal\_config":{"wal\_capacity\_mb":32,"wal\_segments\_ahead":0},"quantization\_config":null,"strict\_mode\_config":{"enabled":true,"unindexed\_filtering\_retrieve":false,"unindexed\_filtering\_update":false}},"payload\_schema":{"Book\_Category":{"data\_type":"keyword","points":6717}}}

My collection info.

### **Key Points**

* Research suggests Qdrant normalizes vectors for COSINE distance, but your scores (e.g., 12.475771) indicate multi-vector aggregation, likely summing maximum similarities.
* It seems likely your setup uses ColBERT-like scoring, where scores exceed 1 due to summing across multiple query vectors.
* The evidence leans toward expected behavior for multi-vector search, not a normalization issue.

### **Direct Answer**

**Understanding Your Collection and Scores**

Your Qdrant collection is configured with COSINE distance for all vector types ("original," "mean\_pooling\_columns," and "mean\_pooling\_rows"), and research suggests Qdrant automatically normalizes vectors for this metric. However, your point scores (e.g., 12.475771) are higher than expected for COSINE similarity, which typically ranges from -1 to 1. This is likely because your setup involves multi-vector search, possibly using a ColBERT-like approach, where scores are aggregated by summing the maximum similarities across multiple query vectors, leading to values greater than 1.

**Why You’re Seeing High Scores**

Given your use of Colqwen embeddings and the "max\_sim" comparator in the configuration, it seems likely that for each query, multiple vectors are generated (e.g., one per token), and the final score is the sum of the highest similarities for each query vector compared to the point’s vectors. This aggregation can result in scores like 12.475771, especially if there are many query vectors, and is expected behavior for such setups.

**What This Means for You**

You don’t need to worry about normalization; Qdrant handles it for COSINE distance. The high scores are likely correct for your multi-vector configuration, reflecting the sum of similarities rather than a single cosine value. To confirm, check how qdrant\_service.reranking\_search\_batch aggregates scores, and ensure your search aligns with this multi-vector approach.

### **Comprehensive Analysis of Vector Normalization and Scoring in Qdrant with Colqwen Embeddings**

This section provides a detailed examination of the user's query regarding unexpected point scores in Qdrant searches using Colqwen embeddings, particularly focusing on vector normalization and distance metrics. The analysis is grounded in the provided collection configuration, code snippets, and related Qdrant documentation, ensuring a thorough understanding for users implementing such systems.

#### **Background on Qdrant and Vector Normalization**

Qdrant is an open-source vector database and search engine designed for efficient similarity search, particularly suited for high-dimensional vector data like embeddings. One critical aspect is the choice of distance metric, which influences how similarity is computed. For COSINE distance, normalization is essential, as it measures the cosine of the angle between vectors, focusing on direction rather than magnitude. Normalization typically involves scaling vectors to unit length (L2 normalization), where the vector's magnitude is 1, allowing COSINE similarity to be computed as a dot product for efficiency.

The documentation for Qdrant, specifically in the "Search" and "Collections" sections

, highlights that for COSINE distance, vectors are automatically normalized during upload. This normalization is performed once per vector when added to the collection, leveraging SIMD optimizations for faster dot product calculations, which is equivalent to COSINE similarity for normalized vectors. This behavior is crucial for maintaining search efficiency and is a standard practice in vector databases for COSINE metrics.

#### **Analysis of Collection Configuration and User Setup**

The user provided their collection configuration, which shows multiple vector configurations: "mean\_pooling\_columns," "mean\_pooling\_rows," and "original," all set to use COSINE distance with a size of 128 and "multivector\_config": {"comparator": "max\_sim"}. The collection has 20,280 indexed vectors and 6,717 points, with 15 segments, indicating a substantial dataset. The payload schema includes a "Book\_Category" field, suggesting the data is categorized, possibly for filtering.

From the user's code, in the ModelService class, the batch\_embed\_query method computes embeddings using Colqwen2 and prints the norms, which are less than 1 (e.g., 0.179, 0.302, etc.), indicating the embeddings are not normalized before being passed to Qdrant. The qdrant\_search\_memory\_tool function then uses these embeddings for a batch search via qdrant\_service.reranking\_search\_batch, and the results show high point scores like 12.475771, 10.883173, etc.

#### **Investigation into Distance Metrics and Scoring**

Given the collection's COSINE distance configuration, the expected behavior is that Qdrant normalizes vectors during upload, and the search scores should be cosine similarities, typically between 0 and 1 for positive alignments. However, the observed scores are much larger, which aligns more with DOT product distance, where scores are the raw dot products and can exceed 1, especially if vectors have large magnitudes.

To explore this, the Qdrant documentation was reviewed, confirming that for COSINE distance, vectors are normalized, and scores should be within -1 to 1. An example from the search documentation showed scores like 0.81, 0.75, etc., within the expected range. This suggests that for COSINE distance, scores should be cosine similarities, but the user's scores indicate a different computation.

Further analysis revealed that the user's setup likely involves multi-vector search, given Colqwen's behavior. Colqwen, similar to ColBERT, produces multiple vectors per query, one for each token or patch. The debug output showed multiple norm values (e.g., 128 values), suggesting that for one query, there are multiple vectors, each of size 128, matching the collection configuration.

In such cases, the search likely computes the similarity between each query vector and the point's vectors, using the "max\_sim" comparator, which takes the maximum similarity for each query vector. Then, the final score is likely the sum of these maximum similarities across all query vectors, a common practice in ColBERT-like models for late interaction scoring. This aggregation can result in scores greater than 1, explaining the user's observations of 12.475771.

#### **Multi-Vector Considerations and Score Aggregation**

The collection's "multivector\_config": {"comparator": "max\_sim"} indicates that for each point, which may have multiple vectors (e.g., original, mean-pooled), the score is the maximum similarity between the query vector and any of the point's vectors. However, with multiple query vectors, the search likely aggregates these scores. An article on multi-vector search

<https://medium.com/%40learn-simplified/advanced-multi-stage-multi-vector-querying-using-the-colbert-approach-in-qdrant-1742f6016391>

discussed combining scores from multiple vectors, sometimes summing them, which could explain high scores if COSINE similarities are summed.

For example, if there are 50 query vectors, each with a maximum similarity of 0.25, summing them could yield 12.5, close to the user's 12.475771. This aligns with the ColBERT approach, where the score is the sum of the maximum similarities for each query token, as detailed in an article on advanced multi-vector querying

#### **Possible Explanations for High Scores**

1. **Multi-Vector Aggregation**: The most likely explanation is that the search involves multiple query vectors, and the final score is the sum of the maximum similarities, leading to values greater than 1. This is expected for ColBERT-like models and aligns with the user's use of Colqwen.
2. **Distance Metric Confirmation**: The collection configuration confirms COSINE distance, and Qdrant normalizes vectors, so the issue is not with normalization but with score aggregation.
3. **Reranking Process**: The reranking\_search\_batch method might involve custom scoring, possibly summing scores across vectors, contributing to the high values.



#### **Recommendations and Verification**

To resolve any remaining concerns, the user should:

* Verify the search method in qdrant\_service.reranking\_search\_batch, ensuring it aligns with multi-vector scoring, possibly summing max similarities.
* Confirm the number of query vectors generated by batch\_embed\_query, which likely outputs multiple vectors per query, contributing to the aggregated score.
* Review Qdrant documentation on multi-vector search<https://qdrant.tech/documentation/concepts/vectors/> to understand how scores are computed for batch searches with multiple vectors.

#### **Conclusion**

Based on the analysis, normalization is done automatically by Qdrant for COSINE distance, as confirmed by the collection configuration and documentation. The high point scores (e.g., 12.475771) are likely due to multi-vector search with a ColBERT-like scoring approach, where the final score is the sum of maximum similarities across multiple query vectors. This is expected behavior and does not indicate a normalization issue.