Qdrant Vector Database

Embeddings, Payloads, and Advanced Filtering

1 Common Embedding Sources

Choosing the right embedding source is a critical decision that balances **cost, performance, and accuracy**.

1.1 1. On-Premise, Optimized: FastEmbed by Qdrant

FastEmbed is optimized for on-premise, high-speed generation with minimal dependencies. It uses quantized weights and ONNX Runtime, making it up to **50% faster** than traditional PyTorch models.

When to choose FastEmbed:

- On-premise execution for privacy-sensitive applications.
- High-speed CPU inference without heavy dependencies.
- Tightly integrated with Qdrant.

FastEmbed Quickstart

```python
from qdrant_client import QdrantClient
from fastembed import TextEmbedding

# Default model: BAAI/bge-small-en-v1.5 (~67MB)
embedding_model = TextEmbedding()
vector = embedding_model.embed("Qdrant is a vector search engine")
```

1.2 2. Managed and Integrated: Cloud Providers

- **Qdrant Cloud Inference:** Managed service directly within your cluster. Eliminates external network latency.
- **Third-Party APIs:** Commercial APIs (OpenAI, Anthropic) offering state-of-the-art models (Trade-off: network latency and costs).

1.3 3. On-Premise, Customizable: Open Source Models

Libraries like `Sentence Transformers` give access to the Hugging Face Hub. Ideal for fine-tuning on domain-specific data or running on local GPUs.

2codebgwhite

primary

Execution	On-premise (CPU/GPU)	Cloud API	On-premise (CPU/GPU)
Speed	Optimized CPU latency	API latency	Varies by hardware
Control	High	Low	Maximum

2 Payloads (Metadata)

Vectors capture semantic relevance, but **Payloads hold structured metadata** for business logic filtering.

2.1 Supported Payload Types

- **Keyword:** Exact string matching (`category: "electronics"`).
- **Integer / Float:** Numerical filtering (`price: 19.99`).
- **Bool:** True/false values.
- **Geo:** Latitude/longitude maps.
- **Datetime:** RFC 3339 format timestamps.
- **UUID:** Memory-efficient ID matching.

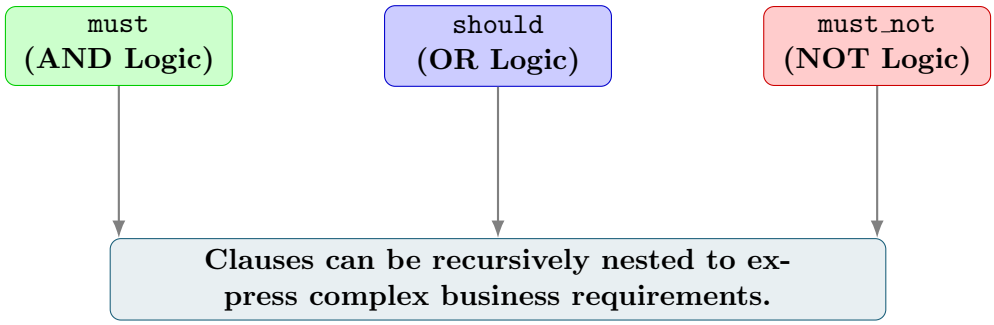
2.2 Data Structures

Payloads can store complex structures:

- **Arrays:** Succeeds if *at least one* value matches (`tags: ["vegan", "organic"]`).
- **Nested Objects:** Queried using dot notation (`user.address.city`).

3 Filtering Logic: Building Complex Queries

3.1 Logical Clauses



Complex Filter Example

```
models.Filter(
    should=[
        models.Filter(must=[
            models.FieldCondition(key="category", match=models.MatchValue(
                value="electronics")),
            models.FieldCondition(key="price", range=models.Range(lt=200))
        ]),
        models.Filter(must=[
            models.FieldCondition(key="category", match=models.MatchValue(
                value="books")),
            models.FieldCondition(key="rating", range=models.Range(gte=4.0))
        ])
    ]
)
```

3.2 Advanced Filtering: Nested Objects

Nested filtering ensures conditions are evaluated within *individual* array elements rather than across all elements.

Nested Condition Example

```
models.Filter(
    must=[
        models.NestedCondition(
            nested=models.Nested(
                key="reviews",
                filter=models.Filter(must=[
                    models.FieldCondition(key="rating", match=models.
                        MatchValue(value=5)),
                    models.FieldCondition(key="verified", match=models.
                        MatchValue(value=True))
                ])
            )
        )
    ]
)
```

4 Performance Optimization

Create payload indexes for frequently filtered fields to maximize performance.

Optimization Note

When filters are highly selective, Qdrant's query planner may bypass vector indexing entirely and use payload indexes for faster results.

Creating Indexes

```
# Index frequently filtered fields
client.create_payload_index(
    collection_name="{collection_name}",
    field_name="category",
    field_schema=models.PayloadSchemaType.KEYWORD,
)

# For multi-tenant applications
client.create_payload_index(
    collection_name="{collection_name}",
    field_name="tenant_id",
    field_schema=models.KeywordIndexParams(type="keyword", is_tenant=True),
)
```

5 Key Takeaways

1. **Flexibility:** Points combine unique IDs, vectors, and structured metadata.
2. **Embedding Strategy:** Balance speed, accuracy, and privacy using FastEmbed, Cloud APIs, or Open Source models.
3. **Powerful Filtering:** Use `must`, `should`, and `must_not` alongside specialized payload indexes to bridge semantic search with strict business rules.