Quantext: QUBO-based RAG for optimal LLM context selection

QD Michaelmas Challenge

Team 01 - Sam Caton, Anton May, Vanja Zdravkovic

The Problem

Explain which problem you are solving. Why do people care about the problem?

LLMs are only as powerful as the context they are provided with. Too little context and they won't have enough information to answer the user's query; too much irrelevant context and they begin to hallucinate. Moreover, a system's hard limits on graphics memory means that is simply impossible to process more than a certain number of raw documents directly with an LLM without expensive hardware improvements.

RAG (Retrieval Augmented Generation) is a field that attempts to fix these issues through a "retriever" which significantly cuts input context to the "generator" (the LLM) while retaining information relevant to the user's query. See the foundational paper "Retrieval-Augmented Generation for Knowledge-Intensive NLP Tasks". The industry-standard method is as follows:

Phase A – Indexing (Offline). User uploads documents, then…

1. **Chunking** – create large set of same-sized contiguous chunks of text from the docs
2. **Chunk embedding** – use pre-trained model to map each chunk to a high-dimensional vector embedding representation, and store these in a vector database $\{v_1, v_2, \ldots, v_n\}$

Phase B – Inference (Online). User creates a query, then…

1. **Query embedding** – create vector embedding for the query, same way as the chunks
2. **Similarity** – find cosine similarity between the query vector and each chunk vector
3. **Top-k selection** – retrieve text from the chunks with the top k similarity values.
4. **Augmented generation** – append this text to the query, and use as an LLM prompt

This already offers significant improvement over passing all documents directly to the generator. However, the approach falls short because of the inherent **lack of diversity** in the retrieved documents. In any research task, there is bound to be symbolically duplicate information in some form, and this increases with the number and size of considered documents. Current RAG systems will naively pass this same information to the generator multiple times, needlessly increasing input tokens and not retrieving important less overt information in the text.

Specifically, we improve upon phase B step 3 to increase information diversity. We propose formulating the retrieval as a QUBO problem, enabling the use of probabilistic computing. Each element of the state represents whether the chunk is included in the retrieved collection. In the objective function, the linear term rewards similarity between the prompt and selected chunks, allowing relevant context to be retrieved. This is balanced by a quadratic term penalising

similarity between chunks, encouraging diversity within the retrieved context. Additional penalties could be imposed to restrict the number of tokens retrieved. Classical solvers are either too slow for real time queries or use approximate methods that produce sub-optimal results whereas a probabilistic computer can rapidly find a high-quality solution.

Let's consider 2 potential areas this could heavily impact:

1. A doctor has an ill patient, and they can't work out the disease. They have access to thousands of large bodies of useful medical text. They want to query, "What illness causes <list_of_observed_symptoms>?". A standard generator-based system fails with a huge amount of context (or provides unreliable info without). A standard RAG-based system retrieves the top k most similar entries, which due to vector similarity are likely to **all be about the same disease**. Our QUBO-based approach instead avoids repetition and retrieves only the required information in a natural and varied set. This means the generator has more access to a **breadth of subject-specific info** – so it becomes trivial to suggest further symptoms to narrow down the disease, and the doctor can work iteratively in his diagnosis.
2. A quantitative analyst is stress-testing a potential multi-million-dollar trade. They have access to years of 10-K filings, earnings call transcripts, analyst reports, and real-time news feeds. They want to query, "What are the key downside risks for <target_asset>?" He will experience similar issues to the doctor: specifically, a direct ChatGPT prompt with all the docs won't work, and ChatGPT via a standard RAG system will likely only retrieve large amounts of info on **a single, obvious risk** (e.g., "inflation concerns"). Our QUBO-RAG system, however, gives a more general and comprehensive review of the asset's risks, making it easier to work out if it can be invested in.

The Market

Current estimates put the size of the RAG market around $2bn, with significant growth expected over the next few years. For example, one report published by MarketsandMarkets estimates that the RAG market will grow from **$1.94bn in 2025 to $9.86bn by 2030** at 38.4% CAGR.

This rapid market expansion is supported by the expected growth in the vector database market, an essential component of RAG systems, from $2.2bn in 2024 to $10.6bn in 2032 (GlobeNewswire). There is already significant investment in key companies in this area like Pinecone (over $100mn raised) and Weaviate (over $50mn raised).

This main driver behind this growth is the need for increased trust in LLM generated responses in key business contexts. Between 2023 and 2025, businesses have invested $12.8bn specifically to address hallucination problems and 78% of leading AI labs consider hallucination one of their top 3 priorities (All About AI).

RAG has emerged the most effective solution to date, cutting hallucination rates by 71% when used properly. Given the amount companies are already dedicating to solving this

problem, further improving the system's performance and reliability represents a highly lucrative opportunity.

Status Quo

Which solutions (if any) are people using currently to solve the problem? Which results will you need to show in order to offer a better solution than what people are currently using? Go into details about algorithms, hardware platforms, etc. if you can. What means success to a customer?

Of course, the foundational problem is *amount of accurate context retrieval*, but for this, since RAG is so standard, we will focus on improvements to the top-k selection method in RAG:

- **Maximal Marginal Relevance (MMR)**
  See "The use of MMR, diversity-based reranking for reordering documents and producing summaries", 1998
  This is the most relevant foundational solution to the *diversity problem* which we are attempting to solve, but it is not state-of-the-art – see learning based reranking below. For this method: instead of simply selecting the top k most similar results, it selects one result at a time, and then performs an algorithm to balance the relevance to the query against similary to selected documents. More specifically MMR selects the chunks using this formula:

  MMR Score = λ * (Relevance_to_Query) - (1 - λ) * (Max_Similarity_to_Results)

  λ controls how much we prioritise relevance compared to diversity. Most industry benchmark papers show MMR achieves no more than 60-75% of the theoretically optimal diversity, and its performance degrades when k>15 or even just when the pool contains strong similarity clusters. Ultimately, for users, this means MMR still misses the important diverse perspectives that a globally optimal solution would present.

- **Learning-based Reranking**
  This approach utilizes neural networks (BERT, or more recent models such as BGE_reranker or Cohere rerank) to score the relevance of these query document pairs more accurately than simple vector similarity. It takes the top k results and rescores them by using the deep bidirectional attention, which often improves the quality of the answer by around 10% on average on standard benchmarks. Companies like Cohere offer reranking as a service at around $1 per 1k searches. However, there are three limitations: It operates only on the initially retrieved set, so if it wasn't retrieved in the first stage, reranking can't introduce diverse information. Second, these models are not trained for diversity, they simply optimize for relevance which promotes redundancy. Finally, the computational cost is large. Cross-encoder inference on 100 candidates adds around 200-500ms latency and thus high cost. For customers this leaves the coverage problem unresolved – if the initial retrieval missed an important piece, reranking can simply not recover it no matter how much precision is improved.

- **Combination with lexical search**
  This is less technical: it just combines the standard semantic vector search with a basic retrieval of documents containing keywords from the query. This hybrid approach (platforms such as Weavlate, Elasticsearch, Pinecone's new hybrid search have implemented for example and they typically use BM25 which is just a probabilistic ranking function) for keyword matching that combines scores using weighted averaging or reciprocal rank fusion is significant. The method helps retrieve the documents that could be semantically distant but lexically very relevant to each other (so for example if we have "aspirin side effects" query it retrieves documents explicitly mentioning "acetylsalicylic acid" even if the embedding similarity is low). This helps us address a different dimension of retrieval failure than the diversity problem directly. The current benchmarks in the field show that this improves recall by around 15% on average on keyword heavy queries whilst adding only minimal latency of around 20ms. Still, it doesn't solve information redundancy, two documents can both match the query keywords whilst being semantically completely identical, and this hybrid search will happily retrieve both which is an issue. For customers, what this means is that hybrid search is simply complementary to diversity optimization and not a substitute and they would still require a robust diversity mechanism to eliminate redundant content.

Solution Structure

Which problems are you looking to implement to offer an adoptable solution (i.e. traveling salesperson, max-SAT, max-CUT, hitting set, restricted Boltzmann machine, ...)?

How can the use case be modelled as a probabilistic computing problem? Which probabilistic algorithms are you proposing to use and which traditional algorithms are you looking to replace? Which benchmarks do you need to run on the QD simulator or hardware to convince customers (give specific metrics for a chosen problem size, problem formulation, required solution quality, time-to-solution, etc)?

What sits between the end-user platform and the probabilistic algorithm? Will the result of the probabilistic algorithm cover the entire output, or do you propose to combine it with other classical algorithms (if the latter is true, how are you deploying the hybrid solution and how much value is generated by the probabilistic part itself)?

Essentially we formulate the diverse retrieval problem as a Quadratic Unconstrained Binary Optimization (QUBO), which belongs to the np hard class of combinatorial optimisation problems. It is very closely related to max CUT but our specific formulation will more directly capture the relevance diversity trade off which is inherent to retrieval and unlike max CUT which tries to essentially partition a graph into two sets (or the travelling salesperson problem for that matter which goes for an ordered sequence), our QUBO seeks an unordered subset of chunks that all together jointly maximize the information coverage.

There is a significant computation challenge (for say 500 candidate chunks and 10 selections there are C(500,10) which is around 2.46 x 10^20 possible subsets to evaluate). Classical

approximate algorithms such as greedy selection (used in MMR) achieve O(nk) time complexity but sacrifice solution quality. Branch and bound exact solvers can guarantee the optimality but have exponential worst case complexity O(2^n), making them really not useful for n>50 so in real time applications. Simulated annealing achieves better quality but requires around 30 seconds for convergence on 500 problems which is just too slow for interactive queries.

This is precisely where probabilistic computing provides a huge decisive advantage, and it explores the exponential solution space in parallel through stochastic dynamics and finding near optimal solutions in milliseconds rather than seconds. The QD probabilistic computer's physical implementation of minimization of energy essentially naturally maps to the QUBO's objective function, which enables it to navigate the complex energy landscape much more efficiently than any of the classical algorithms which simulate the same process in software.

Mathematical Formulation:

Decision variables:

- Binary vector x = (x1, x2, ..., xn) where xi is in {0, 1}
- xi = 1 if chunk i is selected, 0 otherwise

Objective function:

minimize f(x) = -α * Σi sim (query, chunk_i) * xi + (1-α) * Σi Σj>i sim (chunk_i, chunk_j) * xi * xj

subject to: Σi xi = k (select exactly k chunks)

Where:

- Linear term (first sum): Rewards selecting chunks similar to the query (relevance)
- Quadratic term (double sum): Penalizes selecting chunks similar to each other (encourages diversity)
- α parameter: Controls relevance vs. diversity trade off (typically 0.5-0.7)

Integration into Existing Pipeline:

1. Classical Layer (Standard RAG): Handles offline chunking, embedding, and first-stage vector retrieval (Top N candidates).
2. Probabilistic Layer: Uses p-bit hardware to solve the combinatorial QUBO problem (selecting the best k from N).

Benchmarks & Validation:

Target Problem: Selecting k=10-50 diverse chunks from N=500-2000 candidates.

Key Metrics: NDCG (diversity ranking), Subtopic Recall, and Intent-Aware Recall.

Performance Goals:

- Speed: <50ms solve time (beating or matching MMR).
- Quality: 5-15% improvement in alpha-NDCG and 10-20% better subtopic coverage compared to MMR
- Optimality: Within 5-10% of the global optimum.

Validation Datasets: MS MARCO, Natural Questions (Google), and BEIR.

User Journey

Walk through the interaction with the customer and explain how they would access your services.

We have multiple potential avenues/user types we could consider...

Cloud SaaS

Target: Startups and businesses requiring custom rapid deployment. This is the simplest.

- Sign up via web portal (around 5 minutes)
- Upload documents via API or web interface
- All processing done on our servers. This involves 1) context chunking and 2) QUBO offloading to a quantum chip at query time.
- API key provisioning and web playgrounds for testing

Enterprise Deployment

Target: large institutions with data sovereignty requirements, e.g. in healthcare and finance.

This will be easiest done by building around the API solution. But it is our most relevant market.

Setup (around 1 week)

- Customer provides a document corpus (up to 10,000,000 documents)
- We set up system for them via API (specifically we decide on company-specific metrics for diversity and amount of retrieved context)
- We cover benchmarking and fine-tuning to provide them with a custom solution

Integration (around 3 weeks)

- Rest API integration with any existing company LLM infrastructure, or a drop-in-place solution with an existing inference service (possible route for sponsors here)
- Python, javascript, java SDKs
- Potential to handle entire company-specific integration ourselves as a contractor

We would be able to offer a really quick and painless solution to the diversity problem and give people access to a tool they didn't know existed in the first place.