Combined Moodle Search and AI

This document looks at the current Moodle Search implementation to explore if it is feasible to extend this to integrate context indexing to support Retrieval Augmented Generation, with Moodle as a document source.

# Content Indexing

Currently Moodle implements a searching indexing pattern along the lines of Figure 1.

A diagram of a file

AI-generated content may be incorrect.

Figure 1

An indexer script can index the whole site or specific areas, referred to as “contexts” (these are Moodle contexts, not to be confused with an LLM’s context). 2. The indexer script scans the Moodle “database”. This uses a number of objects that understand the Moodle specific structure of the areas (area, activity, block, module) to create a recordset of “documents”. 3. Each “document” object understands how to release an appropriate set of attributes for the search index via the export\_for\_engine() method. 4. This generated data (“index document” in the Figure above) is then passed to the actual search engine (for instance a Solr server) to be stored, and queried against in the future

# Content Retrieval

The corresponding aspect to Content Indexing is Retrieval

A diagram of a computer flow

AI-generated content may be incorrect.

Figure 2

In this process 1. A “search query” is provided by the user (via a web form or other mechanism) 2. This query comprises of some particular context information (such as which parts of Moodle to search) and the user’s actual query terms (“Search Query”) 3. These parameters are passed directly (via the engine layer) to a back end search store (e.g. Solr). 4. The actual search database will then return a set of documents. These are **not** the original assets however, these are the index documents containing metadata about the original items. 5. (Not shown) The Search Engine performs a filter process to check that the result should be returned to the user. 6. These results are passed back, and (typically) rendered into a human readable form, and supplemented within additional garnish, such as a hyperlink to the “original” item. # Context Indexing for RAG In the first instance we can consider content indexing for RAG being a mostly shared process. In fact the biggest difference comes to whether or not there is a “backend” database that Moodle can use for both purposes.

In the first instance we’ll consider the case where there **isn’t** a shareable backend database, we’ll call this “Search *with* RAG content indexing”:

A diagram of a computer process

AI-generated content may be incorrect.

Figure 3

In this case the process is broadly the same as with Content Retrieval above, up to the point where a recordset of content has been fetched and passed back to the search engine.

As before the record set is processed for search, and the resulting document (basically the meta data) is stored in the Search Index (say Solr).

At this point a second process, using the index document and the original document is performed. These are combined and passed to an LLM model for generating an embedding (this is a vector representing the document according to specific LLM). The embedding, metadata and document content is then persisted in *separate* Vector Database (e.g. FAISS, qdrant).

This however is the point where a shared backend could replace these parallel paths:

A diagram of a computer process

AI-generated content may be incorrect.

Figure 4

In this case the backend database is capable of storing both the search index as well as the vector embedding, so only 1 path is necessary.

It should however be noted that a single content “document” could be split in to multiple index documents to fit within size constraints.

# RAG

Retrieval to support RAG is *slightly* different from the search process above.

Specifically we need to provide the LLM with a “context”. This is the extra information that it needs to know, as opposed to a “Moodle context”, which governs the boundaries around content.

A diagram of a computer

AI-generated content may be incorrect.

Figure 5

Much like with a search query, RAG starts with 1. A search query. This is most commonly in the form of free text. 2. The search query is passed to an LLM to generate an embedding. **Note** This function **must** be the same as the one used in the context indexing process, but *doesn’t* have to be the same as the one used to synthesise a response at the end of this process. 3. The Query Embedding vector is then passed to the underlying Vector Database (e.g. Solr, FAISS) which will return a set of result documents determined to be “similar” to the query. 4. These result documents however will also contain the original document’s contents or *parts* of the original document’s contents (ideally the most relevant parts). 5. These results and the original query, can then be fed into an LLM as it’s “context” for the synthesis of a response. 6. The LLM’s response is then returned back out to the user.

Use of Vector Database, Search Database & Search Engine These three 3 terms could be used almost interchangeably, however the “Search Engine” refers to the Moodle layer that normally governs interaction with the underlying database instance (whether it is a Vector or Search one). The Moodle Search Engine is also the only point where anything dependent on Moodle data that can’t be held as static meta-data could be checked.

Retrieval only Whilst this section talks about RAG as a whole process, the Moodle aspect is **only** considering the **Retrieval** aspect (Steps 3 and 4), and how Moodle content could be expose to *any* RAG process. A more complicated process such as Corrective RAG (CRAG) could still be supported by multiple query embeddings being generated in Steps 2 and passed to Step 3. # Conclusion On the basis that accept that Moodle’s current mechanism for correctly identifying content that can be indexed for **search** is suitable for content, then it would look, in theory, practical to extending this feature’s content indexing process so that it not only generates a search index, but either adds vector embedding and content to a dual-purpose backend or maintain that information in a second backend database.

In addition it would then be feasible to expose a Moodle Web Service end point to service AI RAG requests or extend the existing search API to afford extra information for these use cases.

Access Control Finally it is also noted that the **current** Moodle Global Search facility also implements a content filter process *after* the retrieval from the backend data store and *prior* to the return of results, and this process incorporates Moodle security checks and user-visibility checking. It is for this reason that “Moodle” should expose the “Retrieval” service and requests **not** go directly to the underlying database.