Metadata-Enriched PDF Ingestion for Vertex AI Search (Page, Paragraph, Source)"

✅ Acceptance Criteria

1. 📥 PDFs from GCS are Processed Successfully

- All `.pdf` files from the specified GCS bucket are listed and accessed without error.
- A minimum of one document is successfully ingested end-to-end.

2. ✂️ Paragraph-Level Chunking is Accurate

- Each PDF is split into logical paragraphs using layout-aware parsing.
- Each chunk contains the expected `text`, `page number`, `paragraph index`, and `source file`.

3. 🧠 Embeddings are Generated per Chunk

- All paragraph chunks are embedded using `gemini-embedding-001`.
- Embeddings are stored with each chunk but excluded from the final Discovery document upload (for performance).

4. 🧾 Metadata is Properly Injected

- Structured metadata fields (`page`, `paragraph_index`, `source_file`) are added to each Discovery Engine document.

Textual metadata (inline heading) appears at the top of the content for UI display:

css
CopyEdit

```
[Source File: xyz.pdf] [Page: 2] [Paragraph: 4]
```

-

5. 📤 **Chunks are Indexed in Vertex AI Discovery Engine**
    - All enriched chunks are successfully uploaded using `update_document`.
    - Each document appears in the Discovery data store with a unique ID and correct metadata.
6. 🔍 **Semantic Search Returns Valid Results**
    - A query like `"What is retrieval-augmented generation?"` returns top 5 results with relevant content.
    - Each result includes accurate metadata (e.g., correct page and source file from the PDF).
7. 📌 **Fallbacks are Handled Gracefully**
    - If a page has no extractable text, it's skipped without breaking the flow.
    - If `page number` is unavailable, the document is still processed, and the metadata field is omitted cleanly.
8. 🧪 **Verification Step Confirms Document Ingestion**
    - `list_documents()` confirms the correct number of documents in the Discovery Engine data store.
    - Sample document previews show expected structure and metadata.
9. 📈 **Performance is Acceptable**
    - End-to-end ingestion for one standard PDF (10–20 pages) completes in under 1 minute in Colab or dev environment.

Let me know if you'd like these acceptance criteria turned into a **JIRA checklist**, **Confluence table**, or **QA validation form**.

You said:

write 5 comments based on above COnfluence page

ChatGPT said:

Here are **5 detailed JIRA comments** based on the Confluence page you created for the metadata-enriched PDF ingestion pipeline. Each comment aligns with a key implementation milestone and includes a clear summary of what was done.

🗒️ Comment 1: Environment Setup & Project Configuration

✅ Initialized the ingestion notebook with GCP service account authentication and configured the required project settings:

- Set `PROJECT_ID`, `LOCATION_VERTEX`, `BUCKET_NAME`, and `DATASTORE_ID`.
- Verified access to the Cloud Storage bucket and Discovery Engine via SDK.
- Output confirmed with: ✅ `Environment configured. Project ID: rag-agent-project-465118`.

🔧 Prepared dependencies using:

python
CopyEdit

```
os.environ["GOOGLE_APPLICATION_CREDENTIALS"] = "/content/key.json"
```

🗒️ Comment 2: PDF Chunking with Metadata

📄 Implemented logic to extract paragraph-level chunks using `pdfplumber`.

- Each paragraph is tagged with:
    - Page Number
    - Paragraph Index
    - Source File Name

📌 **Output Example:**

json

CopyEdit

```
{
  "text": "This is a paragraph from page 2.",
  "page": 2,
  "paragraph_index": 3,
  "source_file": "sample-doc.pdf"
}
```

This chunking approach ensures each text snippet is small, meaningful, and traceable.

🗒️ Comment 3: Gemini Embedding Integration

🧠 Integrated `gemini-embedding-001` from Vertex AI to generate semantic embeddings for each chunk.

- Chunks were passed in batches to avoid latency.
- Embedding vector stored but excluded from final document push to Discovery Engine (for performance and size optimization).

python
CopyEdit

```
chunk["embedding"] =
embed_model.get_embeddings([chunk["text"]])[0].values
```

Embeddings are essential for enabling high-quality semantic search.

🗒️ Comment 4: Discovery Document Conversion & Upload

📤 Converted enriched chunks to Discovery Engine-compatible documents with inline metadata.

- Metadata stored in both `struct_data` and prepended to document content.
- Used `update_document` to upload each chunk to the configured Discovery datastore.

🎯 Successfully uploaded all documents with unique IDs and verified using `list_documents()`.

🗒️ Comment 5: Semantic Search Validation

🔍 Executed a semantic query:
`"What is retrieval-augmented generation?"`

- Discovery Engine returned top 5 chunks with relevant content.
- Each result included metadata such as:
    - `Page: 4`
    - `Paragraph: 2`
    - `Source File: rrag.pdf`

✅ Verified that the returned results included:

- Correct matching content preview
- Traceable metadata embedded both in structure and raw text
- Fully functional RAG-ready retrieval layer