FABIO AI PROJECT

AI - GENERAL

DETAILED FUNCTIONAL SPECIFICATION - Fabio System

EX-CO	00	21/11/2025	Issued for Comment	Fabio Matricardi	Luigi	Mario	Fabio
Validity Status / Revision Index	Revision Number	Date	Description	Contractor Prepared	Contractor Verified	Contractor Approved	Company Approved

Company logo and business name  ThePoorGPUguy	Project Name FABIO AI PROJECT	Company Document ID **4098FSMDD60232** Job N.
Contractor logo and business name  **Fabio Matricardi**		Contractor Document ID **FAB1-0000-DF-DS-0001** Contract N.
Facility and Sub Facility Name AI - GENERAL		Scale: n.a Sheet of Sheets: 1 / 22
Document Title DETAILED FUNCTIONAL SPECIFICATION - Fabio System		Supersedes N. Superseded by N. Plant Area: n.a. Plant Unit: n.a

File Name: 4098FSMDD60232_EXCO00_Fabio System Detail Specification.doc

REVISION LIST	
Item	**Description**
EX-CO-00	Issued for Comment

HOLD LIST	
No.	**Description**
1	
2	
3	

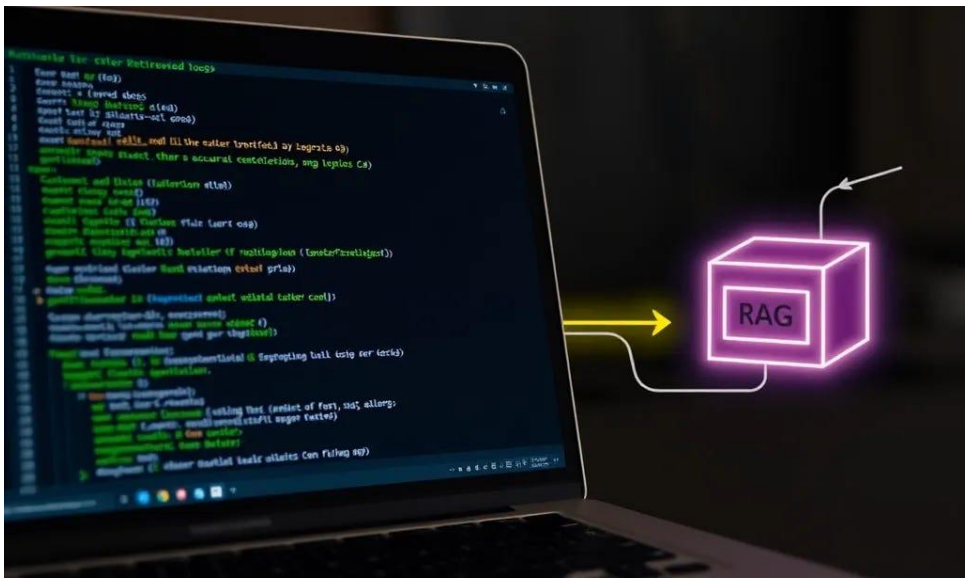
MODIFICATIONS FROM PREVIOUS REVISION		
No.	**Doc Section**	**Modification Description**

TABLE OF CONTENTS

The forgotten Foundation of RAG

1.0 WHY YOUR CHUNKS MATTER MORE THAN YOUR EMBEDDING MODEL?

A few months ago, I spent over $200 running LLM-powered “chunk optimization” experiments.
I fed the same PDF into GraphRAG, LightRAG, and a few LangChain auto-splitters. I tweaked prompts, adjusted overlap windows, even tried letting an LLM rewrite my documents into “RAG-friendly” summaries.
The result?
Worse retrieval. Confusing answers.
And a lot of wasted tokens.
Then I did something radical: I opened the PDF in a text editor, read it carefully, and **drew the chunk boundaries myself**.
Suddenly, retrieval worked. Not “kind of.” Not “with caveats.”
It just… worked.
That’s when I realized: **the biggest bottleneck in RAG is not your model, but your data preparation**.



2.0 THE ILLUSION OF MODEL-CENTRIC RAG

We live in an age where every RAG tutorial starts the same way:
"Just plug in your embedding model and vector database!"
But this skips the most critical step: **making your data retrieval-ready**.
Current frameworks, for all their innovation, often treat documents as inert bags of tokens:

- **LangChain's default splitters** cut at fixed token counts—regardless of whether you're mid-sentence, mid-table, or mid-legal-definition.
- **GraphRAG** (Microsoft's impressive framework) uses LLMs to extract entities and relationships, then builds a knowledge graph. Powerful? Yes. But it requires **dozens of LLM calls per document**, costs scale non-linearly, and the output is opaque… even to its creators.
- **LightRAG** improves on this by using smaller LLM calls to summarize local context, but it still **relies on stochastic reasoning** to decide what belongs together.

All these approaches share a hidden assumption: *that an LLM is the best judge of semantic boundaries*.

- 💡 **But here's a secret: it's not.**

An LLM doesn't know that a footnote belongs to the paragraph above.
It doesn't know that a financial disclaimer applies to the entire section.
It doesn't know that a case study illustrates the principle introduced two pages earlier.
You do.
And that's the core idea behind this series: **the best person to chunk your document is you = the domain expert**.

3.0 WHY CHUNKING IS THE HIDDEN LABOR OF RAG

Retrieval-Augmented Generation (RAG) has three stages:

1. **Retrieval**: find relevant context
2. **Augmentation**: inject that context into a prompt
3. **Generation**: produce an answer

But retrieval only works if the **retrievable units** (your chunks) are meaningful.
Think of chunking like **indexing a book**. A good index doesn't cut entries mid-sentence. It groups concepts logically:
Climate policy → EU regulations, p. 45–48.
A bad index?
Climate, p. 45; policy, p. 46; regulations, p. 47…
… useless.

Yet in RAG, we often hand this critical task to algorithms that have **no understanding of your domain**, your audience, or your intent.
The result?
Chunks that are:

- **Too small**: missing necessary context ("USB-C" without compatibility notes)
- **Too large**: drowning the LLM in irrelevant text
- **Semantically broken**: half a definition + half an unrelated example

And then we blame the embedding model when retrieval fails.

A curious narrative is gaining traction in some corners of the AI discussion: a downplaying of fundamental components like embeddings, cosine similarity, and even the well-established Retrieval Augme…

4.0 INTRODUCING A HUMAN-CENTERED ALTERNATIVE

What if we flipped the script?
Instead of asking, *"How can we make LLMs better at chunking?"*

4.1 We ask: "How can we make it easy for humans to chunk well?"

That's the philosophy behind the toolkit we'll build in this series: a **modular, transparent, and iterative** approach to RAG data preparation.
Our principles:

- **Transparent**: you see every chunk before it's used.
- **Auditable**: every decision is recorded in structured JSON.
- **Efficient**: no wasted LLM calls, no hidden costs.
- **Modular**: each step is a standalone tool you can use today.

Over the next seven posts, we'll build a Gradio application with six tabs:

1. Convert PDFs to clean Markdown
2. Manually set chunk boundaries
3. Inspect your chunks
4. Retrieve with BM25 (yes, keyword search it's still great!)
5. Add local semantic search (embeddings in action)
6. Combine both for hybrid retrieval
7. Reply to a query with hybrid retrieval and with local LLM

By the end, you'll have a full RAG prep and retrieval suite: **that starts with you, not a model**.

4.2 The bigger picture: RAG as a craft, not just an API

We've been sold a vision of RAG as plug-and-play: drop in your data, hit "embed," and get smart answers.
But real-world RAG is more like **bookbinding than button-pushing**. It requires care, judgment, and iteration.
The good news? **You don't need an LLM to do this well.**
You need:

- Clean text (not PDF garbage)
- Structured delimiters (so you can mark boundaries)
- Token awareness (so chunks fit in LLM context)
- A way to inspect and refine

That's it.

And as we'll see in the next post, the first step—converting PDFs into something usable—is where most RAG pipelines fail before they even begin.

Great RAG doesn't start with a prompt. It starts with a paragraph break.

Next week: Why your PDFs are RAG poison—and how Markdown is the antidote.