



DEPARTMENT OF COMPUTER ENGINEERING
SUBJECT: LLM Production and Deployment

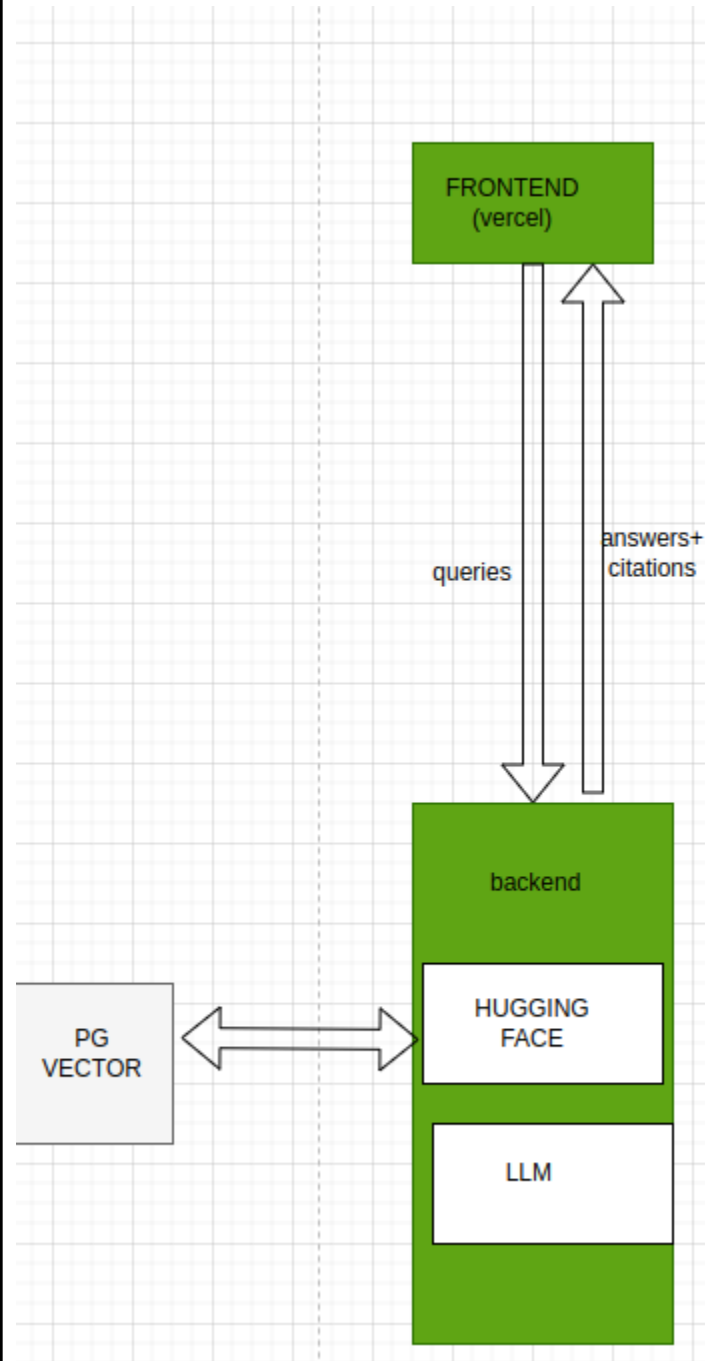
Name	Harsh Shah ,Aamod Sardeshpande
------	--------------------------------

ASSIGNMENT 2	
ABSTRACT :	A full-stack RAG system with Supabase Postgres + pgvector, Gemini 2.5 Flash for answer generation, and HF Inference API for embeddings. Includes chunking ablations, embedding ablations, retrieval evaluation, latency tracking, and a production-ready code. Deployed app returns grounded answers with inline citations and snippet previews.
Design + Diagram:	<p>A production-ready full-stack RAG system using Supabase Postgres + pgvector for retrieval, HF Inference API for embeddings, and Gemini 2.5 Flash for grounded answer generation. Includes chunking/embedding ablations, retrieval evaluation, and latency tracking, with deployment that returns inline-cited answers and snippet previews</p> <p>backend complete view :</p> <pre>graph LR subgraph Hugging_Face_Backend [HUGGING FACE (Backend)] direction TB In[beautiful soup/ playwright] --> Clean[cleaning(remove tags, endlines)] Clean --> Process[process] Process --> Supabase[supabase] Supabase --> Chunking[Chunking] Chunking --> Ablation[Ablation to pick best] Ablation --> Fixed[Fixed(best method)] Fixed --> PG_VECTOR[PG VECTOR] PG_VECTOR --> Recursive[recursive/semantic/fin based/fixd] Recursive --> Ablation end</pre> <p>The diagram illustrates the backend workflow of the RAG system. It starts with a 'beautiful soup/ playwright' scraper that feeds into a 'cleaning(remove tags, endlines)' step, followed by a 'process' step. The processed data is then stored in 'supabase'. From 'supabase', the data flows to 'Chunking', then to 'Ablation to pick best', and finally to 'Fixed(best method)'. The 'Fixed' step outputs to 'PG VECTOR', which then feeds into 'recursive/semantic/fin based/fixd'. There is a feedback loop from 'recursive/semantic/fin based/fixd' back to 'Ablation to pick best'.</p>



DEPARTMENT OF COMPUTER ENGINEERING
SUBJECT: LLM Production and Deployment

overall view:



1. We extract src urls from jio pay website and scrape the entire website and related content

2. this is cleaned and stored in supabase datastore

3. the data is then chunked by different chunking techniques and converted to chunked embeddings in PG vector store

4. The Frontend is Vercel CDN interacting with HF API

5. Every question from user is sent to HF which interacts with vector store and recommends answers with citations from the store via our backend

Data Sources:

Sources:

We used a curated list of public URLs including both static and dynamic



DEPARTMENT OF COMPUTER ENGINEERING
SUBJECT: LLM Production and Deployment

	<p>websites. The same set of URLs was processed through two pipelines (Requests+BeautifulSoup and Playwright) for fair comparison.</p> <p>https://www.jiopay.com/business</p> <p>https://www.jiopay.com/business/privacy-policy</p> <p>https://www.jiopay.com/business/terms-conditions</p> <p>https://www.jiopay.com/business/grievance-redressal-policy</p> <p>https://www.jiopay.com/business/merchant-onboarding-kyc-aml-policy</p> <p>https://www.jiopay.com/business/billpay-terms-and-conditions</p> <p>Coverage: The data span across static and dynamic HTML and JavaScript-rendered pages. For each URL, we extracted cleaned text, token counts, noise ratios, and URL statistics to assess both retrieval success and text quality.</p> <p>Ethics & Compliance: Only publicly accessible, non-sensitive pages were used. Access was respectful, with timeouts to prevent overload. Boilerplate and PII were excluded, and results were stored locally in JSON/CSV for reproducibility without redistributing raw content.</p>
Chunking Ablation:	<p><u>DESIGN</u></p> <ol style="list-style-type: none">1. Structural Chunking : We have used a structural(recursive) chunking strategy , as expected the results have been less than ideal Build chunks by heading hierarchy. We preserve H1/H2/H3 context. Each chunk = H1 > H2 > H3\nparagraph block2. Fixed size chunking : We experimented with more than 9 types from 2 individual configurations : FIXED_SIZES = [256, 512, 1024] FIXED_OVERLAPS = [0, 64, 128] This was best performing.3. Semantic Chunking : We experimented with merge using TF-IDF vector match (from NLP) and experimented with 3 different similarity thresholds 0.25,0.20 ,etc.4. Recursive chunking: A hybrid approach that first applies structural splitting, then semantic sentence merging, and finally falls back to fixed-size splitting for oversized blocks.



DEPARTMENT OF COMPUTER ENGINEERING
SUBJECT: LLM Production and Deployment

5. LLM-based chunking: An external large language model (Gemini 1.5 Flash) is prompted to segment text into almost 300 token coherent topical units, trading efficiency for adaptive quality.

METRICS

1. Semantic Coherence: Measures whether chunk boundaries align with natural semantic units.
2. Completeness: Proportion of queries where all relevant information was captured within retrieved chunks.
3. Size Bandwidth Percentage (size_band_pct): Variability in chunk sizes across the corpus.
4. Information Density (info_density): Ratio of unique content to filler/redundancy per chunk.
5. Domain Grouping NMI (domain_grouping_nmi): Normalized Mutual Information measuring alignment of chunks with domain-specific categories.
6. Throughput: Processing speed measured as documents per second.
7. Weighted Score: Aggregated metric balancing semantic coherence, completeness, and efficiency.

RESULTS



DEPARTMENT OF COMPUTER ENGINEERING
SUBJECT: LLM Production and Deployment

1	strategy	config	#chunks	avg_tokens	time_sec	semantic_coh erence	completeness	size_band_pc t	info_density	domain_grou ping_nmi	throughput	weighted_sco re
2	fixed	fixed_s512_o 64	100	456.44	0.33	0.6297	0.985	0.88	19.1638	0.5119	303.030303	0.737413
3	fixed	fixed_s512_o 0	88	459.28	0.3	0.6042	0.9659	0.8864	19.0108	0.5389	293.333333	0.732031
4	fixed	fixed_s512_o 128	114	463.24	0.38	0.6802	0.9912	0.8684	16.648	0.5022	300	0.721228
5	semantic	semantic_para graph_t512	16	2525.31	0.17	1	0.6875	0.3125	19.2189	0.9003	94.117647	0.624645
6	fixed	fixed_s256_o 128	325	242.51	0.48	0.7561	0.9831	0	16.187	0.4431	677.083333	0.527702
7	fixed	fixed_s256_o 64	217	245.44	0.33	0.6285	0.9885	0	15.1522	0.4213	657.575758	0.486846
8	fixed	fixed_s256_o 0	168	240.67	0.33	0.5293	0.9762	0	17.2019	0.4441	509.090909	0.464757
9	fixed	fixed_s1024_ o128	53	846.79	0.3	0.6995	0.6132	0.1698	17.2407	0.6379	176.666667	0.439918
10	fixed	fixed_s1024_ o64	49	866.88	0.33	0.7063	0.5918	0.1224	17.1009	0.6823	148.484848	0.426444
11	fixed	fixed_s1024_ o0	46	878.46	0.3	0.705	0.6087	0.1304	16.0205	0.644	153.333333	0.415829
12	llm	llm_gemini_fl ash_t300	47	192.4	90.46	0.3967	0.7701	0.1489	21.5512	0.5703	0.519567	0.414043
13	semantic	semantic_sent ence_t512	516	78.3	1.1	0.3863	0.6114	0.031	10.1978	0.4102	469.090909	0.268134
14	recursive	recursive_t51 2	522	78.14	7.63	0.3887	0.6188	0.0441	10.1844	0.3267	68.414155	0.196469
15	structural	structural_ht ml	4	6093.5	65.52	0	0.2737	0	10.2308	1	0.06105	0.150408

The fixed(512 with overlap of 64) chunking strategy worked best .

INSIGHTS

This leads us to believe that all the JIO PAY help pages are actually roughly similar in size (512 tokens) . Moreover, it's likely that each sentence or context for every new point is exactly around 64 tokens .

In other words , it takes me 512 tokens to answer any question /present information about JIO PAY and if you give me 64 tokens of words I can predict the topic we are discussing .

Embedding Ablation :

DESIGN

We evaluate the impact of different sentence embedding models on retrieval performance in a topic-based evaluation setup. Three families of models are tested:

1. MiniLM-L6-v2 (384d): Lightweight, fast transformer encoder optimized for semantic similarity.
2. E5-base-v2 (768d): Mid-sized encoder designed for dense retrieval tasks with explicit query/document formatting.



DEPARTMENT OF COMPUTER ENGINEERING
SUBJECT: LLM Production and Deployment

3. BGE-small-en-v1.5 (384d): Retrieval-focused embedding model optimized for passage ranking and search.

All models are tested under identical conditions: same chunked corpus, cosine similarity scoring, and Recall@5 as the main retrieval metric.

METRICS

1. Recall@5: Proportion of relevant topical chunks retrieved in the top-5.
2. MRR (Mean Reciprocal Rank): Position of the first relevant chunk.
3. Avg Latency (ms): Per-query embedding + retrieval time.
4. Embedding Dimension: Representation size (memory footprint trade-off).
5. Embeddings Size (MB): Storage required for all corpus embeddings.
6. Avg Time/Embed (ms): Time to embed each document in the corpus.
7. Model Size (MB): Disk size of of the model.

RESULTS

Model	Recall@5	MRR	Avg Latency (ms)	Index Size (MB)	Model Size (MB)	Embedding Dim	Embeddings Size (MB)	Avg Time/Embed (ms)	Num Queries
bge small en v1.5	1.667	1.0	10.02	0.61	None	384	0.15	3.44	10
all MiniLM L6 v2	1.633	1.0	19.23	0.61	None	384	0.15	5.51	10
e5 base v2	1.533	1.0	13.32	1.19	None	768	0.29	2.26	10

BGE small has better recall than miniLM,e5 base v2

INSIGHTS

Model choice drives accuracy-efficiency trade-offs: E5-base performs best for relevance but is more expensive in memory and computation. MiniLM is efficient and sufficient for smaller-scale systems. BGE strikes a balance.

Embedding dimension matters: Doubling embedding size (384 → 768) improves semantic recall but increases storage $\sim 2\times$.

Query formatting impacts performance: E5 and BGE require special query

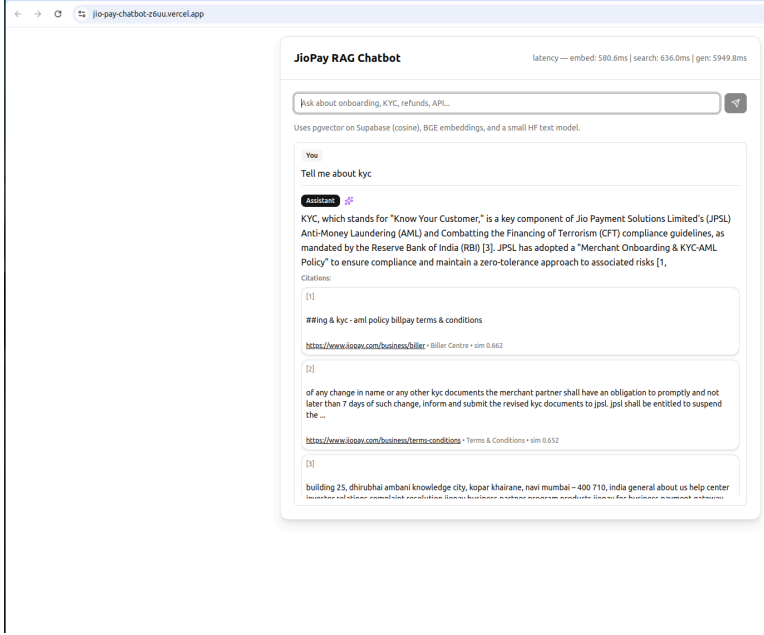


DEPARTMENT OF COMPUTER ENGINEERING
SUBJECT: LLM Production and Deployment

	<p>prefixes to unlock their retrieval capabilities; MiniLM works directly on raw queries.</p> <p>Scalability: For large-scale retrieval, MiniLM or BGE are more practical due to lower embedding storage requirements and faster throughput.</p> <p><u>E5-base-v2 maximizes retrieval accuracy, while MiniLM and BGE-small provide strong trade-offs for efficiency and scalability.</u></p>
Ingestion Ablation	<p><u>DESIGN</u></p> <p>To assess the effectiveness of different web ingestion approaches, we compared two pipelines:</p> <ol style="list-style-type: none">1. Requests+BeautifulSoup (BS4) – a lightweight static HTML fetch and parse pipeline.2. Headless Playwright – a browser-based pipeline capable of executing JavaScript and loading lazy content. <p>Both pipelines used a shared text cleaner to ensure fair comparison: stripping boilerplate tags, collapsing whitespace, and filtering out common navigation or footer tokens. For each URL, we attempted extraction, tokenized the cleaned text, and computed basic quality statistics.</p> <p><u>METRICS</u></p> <p>We evaluated ingestion performance across multiple dimensions:</p> <p>Success Rate: Fraction of URLs with non-empty extracted text.</p> <ol style="list-style-type: none">1. No of Tokens: Aggregate token count across all successfully parsed pages.2. Noise Ratio: Percentage of tokens matching a predefined boilerplate lexicon.3. Throughput: Pages successfully processed per second.4. Failure Rate: Fraction of pages where no text could be extracted.5. Per-URL logs captured characters, token count, and noise ratio, while an aggregate summary table reported average outcomes for each pipeline. <p><u>RESULTS</u></p> <p>Requests+BS4 achieved higher throughput due to low overhead, but failed on pages requiring JavaScript execution, leading to higher failure rates.</p>



DEPARTMENT OF COMPUTER ENGINEERING
SUBJECT: LLM Production and Deployment

	<p>Playwright consistently extracted richer text (greater token counts, lower noise ratios) and captured dynamically loaded content, albeit at significantly lower throughput.</p> <p>Both pipelines produced similar boilerplate filtering efficiency, indicating that the shared text cleaning routine worked as intended.</p> <p>INSIGHTS</p> <p>Static pipelines are preferable when speed is the primary concern, or when targeting websites with well-structured static HTML.</p> <p>Browser-based ingestion is more resilient against adversarial content loading strategies (lazy loads, JS injection), at the cost of computational efficiency.</p> <p>A hybrid approach—defaulting to BS4 with Playwright fallback for failed or high-value pages—could yield the best balance of throughput and completeness.</p>
Retrieval + Generation demo	
Deployment:	<p>The frontend is on VERCEL (content delivery network) so costs are minimal(free for one frontend) and Backend is on Hugging face.</p> <p>INFRA : HF machines + VERCEL machines(AWS + CDN + serverless functions)</p> <p>COST : free for one frontend and minimal queries to backend (rate limit)</p>



**BHARATIYA VIDYA BHAVAN'S
SARDAR PATEL INSTITUTE OF TECHNOLOGY**

Bhavan's Campus, Munshi Nagar, Andheri (West), Mumbai – 400058-India

DEPARTMENT OF COMPUTER ENGINEERING
SUBJECT: LLM Production and Deployment

	MONITORING & SCALABILITY: 1 user at frontend at a time due to Vercel limits , zero monitoring available to developers
Limitations and Future Work :	<p>The limitations include :</p> <ol style="list-style-type: none">1. Lack of multilingual support (hindi queries could be translated to english and then fed to RAG incase of lack of hindi data on JIO PAY this is best option instead of translating entire website to hindi and losing meaning)2. Scalability : Since HF api have rate limits and vehicle allows only 1 deployment this isn't production grade ; instead a deployment to AWS and ASW Sagemaker or other inference providers is suitable3. Performance : TensorRT is used in production grade inference pipelines and model can be converted to optimise performance on A-100 or similar nvidia make GPUs