This is an excellent project. Your core claim is that DeepSeek-OCR's compression makes LLM document processing more *efficient*. Your measurements must therefore prove two things:

1. **Efficiency Gain:** That your method is quantifiably cheaper, faster, or uses fewer tokens.
2. **Effectiveness Parity:** That you don't *lose* critical information in the process of compressing (i.e., the answers are still accurate).

Here are the best tools and techniques to measure this, broken down by what you need to observe.

1. Measuring Efficiency (Cost, Speed, & Compression)

This category measures the "how much better" part of your claim.

Metric to Observe	**Technique / How to Measure**	**Best Tools for the Job**
Token Compression Ratio	This is your project's primary metric. For a batch of documents, you must count: 1. Input Tokens: The token count of the *original, raw text*. 2. Compressed Tokens: The "vision token" count from DeepSeek-OCR's output. **Calculation:** Input Tokens / Compressed Tokens	• **tiktoken (OpenAI):** To count text tokens for GPT models. • **google.generativeai.count_tokens:** To count text tokens for Gemini. • **DeepSeek-OCR Benchmarks:** The official paper provides its own "vision token" counts (e.g., 100 tokens/page) which you should use as your Compressed Tokens baseline.
End-to-End Latency	You need to measure the *entire pipeline time* from (Document Input) -> (Final LLM Answer). This has two parts: 1. T_ocr: Time for DeepSeek-OCR to compress. 2. T_llm: Time for	• **Langfuse** or **LangSmith**: These are perfect for this. You create a "trace" for the whole process. The OCR step is one "span" and the LLM call is a second "span." The dashboard will

	the LLM API call (TTFT + token generation). **Total Latency = T_ocr + T_llm**	automatically show you the latency of each part and the total.
Total Cost per Document	This is the most critical business metric. You need to log the token usage *for every step*. 1. **Your Method:** Cost(DeepSeek-OCR_run) + Cost(LLM_call_with_vision_tokens) 2. **Control Method:** Cost(LLM_call_with_raw_text_tokens)	• **Langfuse / LangSmith**: Both automatically track and sum up token costs for every call within a trace, giving you a precise $ value per run. • **Binadox** or **TrueFoundry**: These are ideal for a production environment. They act as a central gateway to monitor *all* AI spending across *all* models (Gemini, GPT, etc.) and can generate reports showing total cost savings from your compression method.

2. Measuring Effectiveness (Accuracy & Faithfulness)

This is the most important part. A high compression ratio is useless if the final answers are wrong. Your method is a form of Retrieval-Augmented Generation (RAG), where DeepSeek-OCR is the "retriever" and "compressor."

Metric to Observe	**Technique / How to Measure**	**Best Tools for the Job**
OCR Decoding Precision	Before even sending to the LLM, you must verify the compression is faithful. 1. Take a document (ground truth text). 2. Pass it through DeepSeek-OCR. 3. Measure the **Character**	• **jiwer**: A Python library to calculate WER. • **Standard OCR benchmarks** (like Fox or OmniDocBench) mentioned in the DeepSeek-OCR paper, to compare your results

	Error Rate (CER) or **Word Error Rate (WER)** between the decoded text and the ground truth.	against theirs.
Answer Faithfulness	**This is your key quality metric.** Does the LLM's final answer *hallucinate* or invent information that was lost during compression? 1. Ask the LLM a question based on the document. 2. Provide the compressed vision-text as context. 3. Evaluate if the generated answer is *fully supported* by the original document.	• **Ragas**: The best open-source tool for this. You will use the faithfulness metric. • **Langfuse / LangSmith**: You can run Ragas evaluations (or "LLM-as-a-judge" evaluations) directly within these platforms on a dataset of test questions.
Answer Relevancy	Is the LLM's answer actually relevant to the user's question, or did the compression cause it to misunderstand the query or context?	• **Ragas**: Use the answer_relevancy metric.
Contextual Precision	Did the DeepSeek-OCR compression *keep* the most important parts of the document needed to answer the question?	• **Ragas**: Use the context_precision metric.

3. Measuring Resource Consumption (For Self-Hosted OCR)

If you are running DeepSeek-OCR locally, the cost isn't just API calls, it's hardware.

Metric to Observe	**Technique / How to**	**Best Tools for the Job**

	Measure	
GPU VRAM Usage	How much GPU memory is required to run the DeepSeek-OCR (DeepEncoder) model? This determines what hardware you need.	• **nvidia-smi**: The command-line tool. • **GPUtil** or **nvidia-ml-py**: Python libraries to log the peak VRAM usage from within your script.
Compute Bottlenecks	Which part of the vision model is slowest? The SAM component? The compressor? The CLIP component?	• **PyTorch Profiler**: A built-in PyTorch tool that will give you a detailed function-by-function breakdown of where every millisecond is spent on the GPU.

Recommended Workflow & Tool Stack

1. **Local Development (Measuring Quality):**
   - **Goal:** Prove your compressed answers are accurate.
   - **Stack:** Use **Langfuse** (for its open-source, self-hostable nature) and **Ragas**.
   - **Process:**
     1. Create a "golden dataset" of 50 documents with 5 Q&A pairs each.
     2. Run all questions through your pipeline (DeepSeek-OCR -> LLM).
     3. Log all traces (inputs, compressed tokens, outputs, latencies, costs) to **Langfuse**.
     4. Use the **Ragas** integration within Langfuse to score every single run on faithfulness, answer_relevancy, and context_precision.
     5. You now have a dashboard showing the quality and cost for your entire test set.
2. **Production (Measuring Cost & Governance):**
   - **Goal:** Monitor and control your total AI spend across Gemini, GPT, and your self-hosted model.
   - **Stack:** Use **TrueFoundry** or **Binadox**.
   - **Process:**
     1. Route all API calls (both to OpenAI/Google and your internal DeepSeek model) through the **TrueFoundry** AI Gateway.
     2. Use the **Binadox** dashboard to get a simple, high-level view of your total token consumption and cost, broken down by model and team.

3. Set budgets and alerts in Binadox to get notified *before* your costs spike, ensuring the efficiency you promised is being delivered.