# Local LLM Benchmarking (Llama 3.2 3B, Qwen 2.5 3B, Gemma 3 4B)

This repository contains scripts and data to benchmark lightweight local Large Language Models (LLMs) on on-premises hardware using an **Ollama** backend with a PowerShell wrapper and Python script. We evaluate three models – **Llama 3.2 (3B)**, **Qwen 2.5 (3B)**, and **Gemma 3 (4B)** – measuring their **throughput**, **factual accuracy**, and a combined **composite score**. The goal is to determine if small, self-hosted models can deliver acceptable performance and accuracy for tasks (in this case, Q&A about Shram.ai’s mission), and to identify which model is best suited for different hardware tiers.

## Table of Contents

* [Overview](#overview)
* [Models Used](#models-used)
* [Methodology](#methodology)
* [Folder Structure](#folder-structure)
* [How to Run](#how-to-run)
* [Sample Results](#sample-results)
* [Key Insights](#key-insights)
* [Optional Enhancements](#optional-enhancements)

## Overview

**Project Purpose:** This project provides a systematic benchmarking of local LLMs to gauge their viability as an alternative to using large cloud-hosted models. By running models locally, we aim to gain benefits in privacy (data stays on-premise), cost (no API fees), and control (ability to fine-tune and integrate deeply). The benchmark focuses on: - **Throughput & Latency:** How fast can each model generate text on local hardware (tokens per second, and time per response)? - **Factuality:** How accurate are the model’s answers to factual questions, measured against a ground-truth dataset? - **Overall Utility:** Using a composite metric that balances speed and accuracy, which model and settings give the best trade-off on various hardware?

**Benchmark Scenario:** We simulate a Q&A scenario relevant to **Shram.io’s mission**. Each model is asked to “*Write a brief story about Shram’s mission and its impact on work using the provided context.*” with context snippets given (mimicking a retrieval-augmented prompt). The models’ responses are compared to reference answers to assess factual accuracy. We test each model under multiple **sampling temperatures** (0.0 for deterministic, 0.7 for moderate creativity, 1.0 for high creativity) to observe how creativity vs. factuality trade-offs affect performance.

Multiple machines of varying specs (from older 4GB VRAM GPUs up to modern 12–16GB VRAM GPUs, and even a CPU-only scenario) were used to see how model performance scales with hardware. The included scripts allow you to reproduce the benchmarks on your own Windows machine (or adapt for other OS) and examine the results.

## Models Used

We benchmark the following **open-source LLMs** (served via [Ollama](https://github.com/jmorganca/ollama) for easy local inference):

* **Llama 3.2 – 3B parameters** (Meta AI): A 3-billion-parameter model (third major version of Llama) known for strong general knowledge even at small scale. *(Chosen for its high accuracy relative to size.)*
* **Qwen 2.5 – 3B parameters** (Alibaba Qwen): A 3B model from the Qwen series (version 2.5) optimized for efficiency. *(Chosen for its balance of speed and capability, and to represent a different architecture.)*
* **Gemma 3 – 4B parameters** (Open source model): A 4-billion-parameter model (third iteration) with slightly larger size. *(Chosen to see if a somewhat larger model offers accuracy gains to justify its heavier footprint.)*

All three models are relatively small (3–4B), making them feasible to run on consumer-grade GPUs. They were the most compact models available in mid-2025 on the Ollama platform for their respective model families. This selection allows testing different architectures while keeping hardware requirements reasonable.

## Methodology

**1. Benchmark Script:** The core benchmarking is done by benchmark\_extended.py, which performs a series of tests for each model and temperature: - **Throughput Test:** For each model at each temperature, the script prompts the model once (with a fixed-length prompt and context) to generate a response of up to a certain number of tokens (e.g. 256 tokens). It measures the time taken and calculates **tokens per second** (and per minute) as well as the **latency** for the response. GPU memory usage (VRAM before/after) is also logged. This gauges how fast the model runs on the given hardware. - **Factuality Test:** The script can load a factual Q&A dataset (by default shram\_ai\_dataset.json, containing 10 factual questions about Shram’s mission and product). For each question, and for each model-temperature combination, the model generates a short answer. The answer is then scored against the reference answer using a simple token overlap F1 score (measuring precision and recall of factual terms). We average these scores across all questions to get an overall **factual accuracy percentage** for that model at that setting. - **Composite Score:** We combine the normalized throughput and factuality into one **composite score** to reflect overall performance. After all tests, the script normalizes the efficiency (throughput) and factuality values to [0,1] range (where 1.0 is the best among the tested combos on that machine). Then it computes Composite = 0.6 \* Factuality + 0.4 \* Efficiency (by default, giving factual accuracy 60% weight and speed 40% weight). This composite metric helps identify which model+setting best balances being fast **and** accurate. The highest composite score indicates the optimal choice on that hardware.

**2. Temperature Settings:** We run each model at three temperature settings – **0.0 (deterministic)**, **0.7 (moderate)**, and **1.0 (highly random)**. Lower temperatures typically yield more factual but sometimes stale responses, while higher temperatures increase creativity at the risk of factual errors. By testing all three, we observe how each model’s accuracy and diversity change, and we can pick the best setting for each model. Notably, changing temperature **does not significantly impact throughput** (since the computational cost per token remains similar), which was confirmed by our results.

**3. Hardware Variations:** The benchmark was executed on multiple systems: - **Low-end GPU:** e.g. NVIDIA GTX 1650 (4GB VRAM). - **Mid-range GPU:** e.g. NVIDIA RTX 2060 SUPER (8GB VRAM). - **High-end GPU:** e.g. NVIDIA RTX 4070 Ti or similar (12–16GB VRAM). - **CPU-only:** (Optional) by running Ollama with --no-gpu, to observe performance without a GPU.

Each run produces a JSON file with detailed metrics (and the script captures the hardware info like CPU, GPU name, RAM). By aggregating results across machines, we can see how performance scales. In general, GPU memory and compute power had a huge effect on throughput (small models on a strong GPU can be an order of magnitude faster than on a weak GPU), while factual accuracy remained consistent across hardware (since that’s model-dependent, not hardware-dependent, except when severe memory limits forced some CPU fallback).

**4. Dataset:** The factual accuracy was evaluated with **10 questions** about Shram’s mission, features, and impact. These are stored in shram\_ai\_dataset.json. Each entry has a prompt (the question or instruction) and a list of acceptable answers (reference facts). The prompt is fed to the model along with a relevant context snippet (the script automatically prepends context from the dataset to simulate an open-book scenario). The model’s answer is then compared to the reference answers for overlap of key facts. This approach approximates factual correctness (an F1 score of 1.0 would mean the model output all relevant facts perfectly). We focus on factual correctness in a domain-specific context because our intended use-case is an internal Q&A assistant for Shram.

**5. Metrics Collected:** For each model at each temperature on each hardware, we record: - *Tokens per second* (throughput) and *latency per prompt* (seconds to generate ~256 tokens) – higher throughput / lower latency is better. - *Factuality score* (average F1 overlap with references) – higher is better. - *Normalized efficiency and factuality* (relative to best in that run) and the *composite score* (0 to 1 scale) as described above. - *Model diversity* (an optional metric indicating how much the model’s outputs vary at different temperatures, measured by unique token usage – higher diversity typically at higher temperatures, though we gave it less importance in this factual benchmark).

All these metrics are output in the JSON results for analysis. The script also prints a summary table to console at the end of a run, showing each model’s metrics per temperature.

## Folder Structure

The repository is organized as follows:

* **benchmark\_extended.py** – The Python script that conducts the benchmark. It can be run on any platform (requires Python 3 and the packages requests, psutil, and optionally pandas). Contains detailed documentation at the top explaining usage and methodology.
* **run\_extended\_benchmark.ps1** – A PowerShell script for Windows to conveniently run the benchmark with preset parameters. This script sets the model list, temperature list, prompt, dataset path, and composite weights, then calls benchmark\_extended.py. It by default runs a GPU inference test (assuming Ollama is serving on localhost:11434). *(Windows users can execute this script to run the entire benchmark in one go.)*
* **run\_extended\_benchmark.sh** – *(If using Linux/MacOS)* A shell script equivalent to the PowerShell script, allowing non-Windows users to run the benchmark with similar settings. Make sure to edit executable permissions (chmod +x) and adjust the Ollama server URL if needed.
* **shram\_ai\_dataset.json** – The factual Q&A dataset (Shram mission questions) used for evaluating factual accuracy. It’s a JSON with a top-level "questions" list containing objects with "prompt" and "answers".
* **Results/ directory** – Contains sample output files from benchmark runs on various machines. For example:
* extended\_results\_<HostName>\_GPU.json – JSON results from a GPU run on a machine (where <HostName> is the computer name). It includes the hardware specs and an array of entries for each model & temperature combination with their metrics and even the generated response text.
* extended\_results\_<HostName>\_CPU.json – JSON results from a CPU-only run (if performed) on that machine.
* *(You can open these JSON files to inspect detailed metrics or use them for further analysis/plotting.)*
* *(Optional)* **Plot Images:** While not included by default, if you generate graphs from the results, you can place them here for reference:
* throughput\_by\_model\_temp.png – A plot of model throughput (tokens/sec) for each temperature.
* factuality\_by\_model\_temp.png – A plot of average factual accuracy (F1) for each model at each temperature.
* composite\_score\_comparison.png – A bar chart comparing the composite score of each model at its optimal temperature setting.
* *These images are referenced in the analysis discussion (Figures 1, 2, 3) and can be added to the README or repository if needed for visualization.*

## How to Run

Follow these steps to run the benchmark on your local machine (Windows instructions using PowerShell, with notes for other OS):

1. **Prerequisites:**
2. Install **Python 3.x** and ensure pip is available.
3. Install required Python packages:

* pip install requests psutil pandas
* (Note: pandas is optional, used only for pretty-printing results. The script will run without it, but installing it is recommended for the formatted console output.)

1. Install **Ollama** (the LLM runtime) from [Ollama’s releases](https://github.com/jmorganca/ollama). Ollama allows you to run models locally via an API.
2. Download or **pull the models** into Ollama. Make sure the model tags in the script match what you have:

* ollama pull llama3.2:3b  
  ollama pull qwen2.5:3b  
  ollama pull gemma3:4b
* *(These commands will download the models if available. If the exact tags are unavailable, use the closest equivalent models that are installed and update the script’s $models list accordingly.)*

1. **Hardware**: If you plan to run a GPU and a CPU test, note that you might need to start a separate Ollama instance with CPU-only mode. See step 3.
2. **Start the Ollama server:**  
   Ensure Ollama’s daemon is running. By default, Ollama listens on http://localhost:11434. You can start it by running:

* ollama serve
* This will launch the local model-serving engine (using GPU if available). If you want to perform a CPU-only benchmark as well, you can start a second instance on a different port (e.g., 11435) with GPU disabled:
* ollama serve --no-gpu --port 11435
* In the PowerShell script, the base URL for GPU is set to localhost:11434. You could modify the script or run a second pass for CPU by changing the base URL and output suffix accordingly.

1. **Edit settings (optional):** Open run\_extended\_benchmark.ps1 in a text editor if you want to adjust the parameters:
2. $models: list of model names/tags to test. By default it includes gemma3:4b, qwen2.5:3b, llama3.2:3b. Ensure these match models you have in Ollama (ollama ls can list installed models).
3. $temperatures: sampling temperatures to test (default 0.0, 0.7, 1.0).
4. $numPredict: number of tokens to generate per prompt (default 256).
5. $prompt: the prompt text for generation (a general instruction about writing a story on Shram’s mission; this gets combined with context in the Python script).
6. $factsPath: path to the factuality dataset JSON (default shram\_ai\_dataset.json in the same directory).
7. $weightFact and $weightEff: weights for factuality vs efficiency in the composite score (default 0.6 and 0.4 respectively).
8. Output filenames are auto-generated as extended\_results\_<COMPUTERNAME>\_GPU.json etc., you can change the naming or directory if desired.
9. **Run the benchmark:**
10. **On Windows:** open PowerShell in this repository’s folder and execute:

* .\run\_extended\_benchmark.ps1
* This will launch the Python script with the configured parameters. It will take some time as it runs multiple prompts and questions through each model. Progress and status messages will be printed in the console. Upon completion, you should see a message indicating the results have been saved (e.g., extended\_results\_MYPC\_GPU.json created).

1. *(Optional CPU run):* If you started the CPU-only Ollama on port 11435 and want to benchmark CPU performance, you can modify the script to call Run-Bench 'http://localhost:11435' 'CPU' (similar to how GPU is run). For example, you could duplicate the last line in the PS script and adjust it for CPU. Then run the script again to produce a \_CPU.json result. (Be aware that CPU inference will be **much** slower.)
2. **On Linux/Mac:** you can use the provided run\_extended\_benchmark.sh in a similar way:

* chmod +x run\_extended\_benchmark.sh  
  ./run\_extended\_benchmark.sh
* (Ensure the script’s paths and ports match your setup.)  
  Alternatively, you can run the Python script directly with your desired arguments. For example:
* python benchmark\_extended.py \  
   --base-url http://localhost:11434 \  
   --models llama3.2:3b qwen2.5:3b gemma3:4b \  
   --temperatures 0.0 0.7 1.0 \  
   --prompt "Write a brief story about Shram's mission and its impact on work using the provided context." \  
   --num-predict 256 \  
   --factuality-dataset shram\_ai\_dataset.json \  
   --weight-factuality 0.6 --weight-efficiency 0.4 \  
   --output extended\_results\_myhost\_GPU.json
* *(You can adjust models, temperatures, etc., as needed. Running directly allows flexibility if you are not on Windows.)*

1. **View Results:** After running, find the output JSON file in the directory (or in the Results/ folder if configured). You can open it in any JSON viewer. It contains a top-level "hardware" section with system info and an "entries" list with each test combination. Each entry includes the model name, temperature, and all the metrics (throughput, latency, VRAM usage, factuality scores, composite, etc.), as well as the model’s generated response text for the prompt. This data can be further analyzed or visualized. A quick summary was also printed to the console. For a more readable report, you might consider writing a small script or using Jupyter Notebook to parse these JSONs and produce charts (some example plots are discussed in the next section).

## Sample Results

After running the benchmarks on multiple systems, we observed clear patterns in performance. Below is a summary of key results for each model:

* **Throughput (Speed):** On a mid-range GPU (8GB, e.g. RTX 2060 Super), the **Llama 3.2 (3B)** model generated about *~66 tokens per second*, **Qwen 2.5 (3B)** about *~62 tokens/sec*, and **Gemma 3 (4B)** around *~55 tokens/sec*. On a high-end GPU (12GB+ like an RTX 4070 Ti class), throughput increased dramatically: Llama 3B exceeded *150 tokens/sec*, Qwen 3B reached around *140 tokens/sec*, and Gemma 4B around *120 tokens/sec*. On a very low-end 4GB GPU, throughput dropped significantly (Llama and Qwen in the 20–25 tokens/sec range, and Gemma sometimes as low as single-digit tokens/sec if it had to swap to CPU memory). **Temperature settings had minimal impact on throughput** – the lines for 0.0, 0.7, 1.0 were virtually flat in our tests (e.g., Llama 3B might be 150 vs 148 tokens/sec between 0.0 and 1.0, within margin of error). This means you can choose a sampling temperature based on desired output quality without worrying about speed loss.
* **Factual Accuracy:** We report the average F1 overlap score (0 to 1) against reference answers. **Llama 3.2 (3B)** was the most factual model overall, achieving about **0.65 F1** (out of 1.0) at temperature 0 (greedy deterministic). **Qwen 2.5 (3B)** was a close second with around **0.60 F1** at T=0. **Gemma 3 (4B)** trailed with around **0.56 F1**. These scores indicate that on average the models’ answers contained roughly 56–65% of the key facts from the references (not perfect, but decent for 3B-size models on a niche topic). As expected, **higher temperature led to lower factuality**: at T=1.0, all models dropped in accuracy (Llama fell to ~0.62, Qwen dropped more significantly to ~0.52, Gemma around ~0.54). Interestingly, Qwen showed a tiny improvement from T=0 to T=0.7 (0.60 -> ~0.61) before falling off, suggesting a little randomness helped in some cases, but too much hurt. Llama and Gemma steadily declined with more randomness. Bottom line: for maximum factual precision, use deterministic or low temperatures; moderate temperature can sometimes help phrasing but has a small cost in accuracy.
* **Composite Score:** Combining speed and accuracy into one metric (with 60% weight on factuality, 40% on efficiency) helps identify the best overall performer on each hardware. In our tests, **Llama 3.2 (3B)** almost always came out on top with the highest composite score. For instance, on a powerful GPU, Llama at T=0.7 achieved a composite score near **0.99** (on a 0–1 normalized scale, essentially nearly optimal). **Qwen 2.5 (3B)** also scored high, typically just behind Llama – e.g. around **0.90–0.93** on the same scale. **Gemma 3 (4B)** lagged, with best composite around **0.80–0.85**. It’s worth noting that on the weakest hardware (e.g., a 4GB GTX 1650), Qwen 3B actually edged out Llama in composite score because Qwen maintained better speed under tight memory constraints, whereas Llama slowed down more. (For example, on that 4GB GPU, Qwen at T=0 had the highest composite among the combos, due to being ~2x faster than Llama there, even though Llama was slightly more accurate.) However, on any mid-range or better GPU where all models can run well, Llama’s superior accuracy keeps it at the top. Gemma never achieved the top composite on any machine – its extra parameters didn’t translate to better factuality, so it was generally a net negative in our balanced metric.

*Visualization:* If you have the plot image files available in the repo, you can refer to them for a quick visual summary of these results: - **Figure 1** – *Throughput vs Temperature:* (throughput\_by\_model\_temp.png) shows each model’s tokens/sec at T=0.0, 0.7, 1.0. You’ll see Llama and Qwen significantly above Gemma, and nearly flat lines across temperatures. - **Figure 2** – *Factual Accuracy vs Temperature:* (factuality\_by\_model\_temp.png) shows the average F1 score for each model at each temp. Llama is highest at all points, Qwen slightly below, Gemma lowest; all decline as temp increases (especially Qwen at T=1.0). - **Figure 3** – *Composite Score Comparison:* (composite\_score\_comparison.png) presents each model’s composite score at its optimal temperature (the highest point from figures 1 & 2 combined). Llama’s bar is tallest overall, Qwen’s just a bit shorter, and Gemma’s noticeably lower. On some low-end cases Qwen’s bar could match or exceed Llama’s, but in aggregate Llama leads.

*(If the images are not embedded here, you can generate or view them by processing the JSON results. They illustrate the numeric points discussed above.)*

## Key Insights

Based on the benchmarking results, here are the key takeaways and recommendations for choosing a model depending on your hardware:

* **🥇 Best Overall Model – Llama 3.2 (3B):** Across the board, Llama 3B provided the best mix of accuracy and speed. It has the highest factuality scores and is extremely fast on any decent GPU. For **mid-range (6–8GB VRAM)** and **high-end GPUs (≥10GB)**, **Llama 3B is the recommended model**. On an 8GB card it runs comfortably and outputs more accurate answers than others (even if Qwen might be a couple tokens/sec faster, the difference is minor). On high-end GPUs, all models run very fast, so Llama’s accuracy advantage makes it the clear winner for quality. It is the ideal choice for an on-premises Q&A assistant if your hardware can support it.
* **🥈 Second Choice – Qwen 2.5 (3B):** Qwen 3B is not far behind Llama. It actually excels on **lower-end hardware**. For **low VRAM GPUs (~4GB)** or situations with limited memory, **Qwen 3B is the safest choice** among the three. It has a smaller memory footprint and remained reasonably fast and moderately accurate when Llama struggled or had to fall back to CPU. In our tests, Qwen outperformed Llama on a 4GB GPU in composite score due to much higher speed, even though its answers were slightly less accurate. If you only have a very low-end GPU (or are running on CPU-only), you might even consider using an even smaller model than 3B; but from our tested set, Qwen 3B is the most viable on constrained devices. On better hardware, Qwen still does well – it’s a viable backup to Llama, especially if future tasks require strengths that Qwen might have (some reports suggest Qwen models are strong in certain reasoning or coding tasks for their size). In summary, **use Qwen 3B for hardware-constrained scenarios or as a fallback**.
* **🥉 Gemma 3 (4B) – Not Recommended (in this setup):** Despite having more parameters, Gemma 4B did not outperform the 3B models in factuality. Its answers were consistently less accurate than Llama’s (and slightly below Qwen’s), and it is heavier to run (needs more VRAM and was slower, especially on mid/low hardware). We did not observe any advantage that would justify using Gemma over the other two. Unless a future version or specific domain test shows Gemma excelling in something, **we do not recommend Gemma 3 (4B) for this use case**. It simply lags in the speed-accuracy balance.
* **Impact of Temperature:** The optimal temperature setting for factual Q&A was around **0.0 to 0.7**. For all models, **deterministic mode (0.0)** gave the highest factual accuracy, but slightly lower composite scores because some randomness can improve answer phrasing or brevity which affects efficiency. A **moderate temperature (0.7)** offered a good trade-off – it introduced a bit of variation and in Qwen’s case even improved factual score slightly, and it’s where each model tended to reach its peak composite score. **High creativity (1.0)** is not ideal for factual tasks, as it led to more omissions or hallucinations (especially noticeable with Qwen). So, if using these models for similar tasks, we’d suggest staying in the 0–0.7 range. You can start deterministic for maximum correctness and only raise temperature if you need the model to be less verbosely correct or more varied in output.
* **Hardware Matters (Throughput Scaling):** One of the stark findings was how much **GPU capability** influences throughput. A modern GPU with sufficient VRAM can generate tokens 5–10x faster than an old or low-memory GPU for these models. In practical terms:
* On a high-end card (RTX 4070 Ti class), answers of a few hundred tokens were generated in under 2 seconds. This means a local LLM can feel interactive and snappy.
* On a mid-tier 8GB card, that same answer might take ~3–5 seconds, which is still quite usable for many applications (a short wait).
* On a 4GB or older GPU, the wait could be 10+ seconds, which starts to feel sluggish. And in worst cases (if the model doesn’t fully fit in VRAM), the system might spill to CPU and slow to a crawl (as we saw with Gemma on 4GB, taking 30+ seconds).
* **CPU-only** inference is possible but *very slow* – these 3B models would take on the order of minutes per response on a CPU, so we consider that only for testing or if absolutely no GPU is available. In practice, even a low-end GPU acceleration is hugely beneficial.

Therefore, for deployment, it’s important to match the model choice to your hardware. The benchmarks suggest: - *4GB VRAM:* Use the smallest model (Qwen 3B) or even consider a 1–2B model not in this test, to avoid memory swapping. - *6–8GB VRAM:* Llama 3B runs fine; you get decent speed and best accuracy. - *>10GB VRAM:* All tested models run at high speed; go with Llama 3B for best results.

Also, GPU generation was found to be largely **GPU-bound**; differences in CPU (AMD vs Intel, clock speeds) on the host didn’t significantly affect things as long as the GPU was handling the model. Consistent software (all tests used Ollama on Windows) ensured comparable conditions.

* **Viability of Local Models:** These results show that small local models **can** serve as a foundation for an in-house Q&A or assistant system. They are not as **accurate** as large cloud models (for perspective, GPT-4 would likely score near 1.0 on these factual questions), but ~60% factual accuracy is a starting point that can be improved by providing good context (our test already gave some context) or fine-tuning. The speed on a good GPU is actually on par with or better than calling an API (no network latency and fully in parallel for multiple queries if you have the hardware). Importantly, running locally means **zero cost per query** and **full data privacy** – crucial for sensitive internal usage. We must be mindful of the limitations (they can miss details or make errors that a larger model wouldn’t), but for many straightforward queries or with an added retrieval system, these 3B models are surprisingly capable.
* **Recommendations:** For our specific use case (answering questions about Shram.io internally), **Llama 3.2 (3B) at a moderate temperature (~0.5–0.7) with context injection is the chosen solution**. Qwen 3B remains an excellent backup or secondary option (and we will monitor its performance on other tasks). We would avoid using Gemma 4B given the outcomes. We plan to integrate a documentation knowledge base (vector store) so that the model always has relevant info to draw on – this should boost factual accuracy significantly (potentially from ~60% to close to 100% for answerable questions). The composite scoring approach here helped confirm that, as long as we have at least a mid-tier GPU, Llama 3B doesn’t force a trade-off between speed and accuracy – it gave the best of both, which is very encouraging.

In summary, **small local LLMs are viable** for our needs, and the best model for most scenarios tested is Llama 3.2 (3B). Qwen 2.5 (3B) is nearly as good and especially useful when resources are limited. Gemma 3 (4B) is generally not worth the extra size. With the right model and hardware, you can achieve fast, reasonably accurate responses without relying on external AI services.

## Optional Enhancements

This benchmarking project can be extended or improved in several ways. Some ideas for future enhancements:

* **Test Additional Models:** Include more models or newer versions. For example, when **Llama 3.3** or other small models (like GPT-3.5 2.7B variants, MPT, etc.) become available, you could add them to the comparison. It would be interesting to see if any 1–2B parameter models can approach the performance of these 3B ones on such domain-specific tasks, or if a 7B model yields a big jump on a high-end GPU.
* **Automate Multi-Machine Merging:** Currently, we ran the script on multiple machines and manually aggregated the results for analysis. A useful enhancement would be a script or notebook to **merge JSON results** from different hosts and produce summary tables/plots automatically. This could involve combining the "entries" from each JSON and computing global stats like average performance and variability across hardware.
* **Visualization Integration:** Add a step in the Python script or provide a Jupyter Notebook to generate the plots (throughput vs temperature, factuality vs temperature, composite scores, etc.) directly from the results. This would make it easy for users to see charts of their own hardware’s performance. The example images mentioned (throughput\_by\_model\_temp.png, etc.) could be produced programmatically.
* **More Metrics:** We focused on factual accuracy and speed. Other evaluation axes could be added:
* **Quality/Fluency**: e.g., using BLEU or human ratings to judge answer coherence.
* **Memory Usage**: beyond just peak VRAM, track system memory or any initialization times.
* **Stability**: run multiple trials to measure variance in outputs (especially at higher temperatures).
* **Prompt Generality**: test with a variety of prompt types (not just the Shram context) to see if results hold for other content.
* **Diversity Index**: we calculated a diversity score (unique token usage) – this could be explored more if creative generation is a focus.
* **Extended Dataset:** Use a larger or different factuality benchmark, such as **TruthfulQA** or domain-specific FAQs, to further validate model accuracy. Our dataset was small (10 questions); expanding it or using standard benchmarks would give more confidence in the results.
* **Fine-tuning and Retraining:** Experiment with fine-tuning these models on your specific data (if allowed by model license) to see if factual accuracy can be improved. For example, fine-tuning Llama 3B on Shram’s documentation might raise that 0.65 F1 closer to 0.9+. This repository could be extended with a fine-tuning script or guidelines.
* **Integration with Retrieval**: As mentioned, coupling the model with a retrieval system (providing relevant documents as part of the prompt) can greatly enhance factual performance. An optional enhancement is to integrate a small pipeline that fetches context from a knowledge base before the model answers. Benchmarking *with* and *without* retrieval would be an informative extension.
* **Support Other Inference Backends:** While we used Ollama for convenience (which internally uses something like llama.cpp or similar for model execution), one could test the same models on different runtimes or libraries (e.g., Hugging Face Transformers, GPTQ, TensorRT, etc.) to compare performance. A wrapper to benchmark across backends could be developed.
* **Continuous Benchmarking:** If this is for a product, setting up a continuous benchmark (CI pipeline or scheduled job) to re-run these tests when models or software updates occur could catch regressions or improvements over time. For example, as new versions of Ollama or GPU drivers come out, you might see changes in throughput.

Feel free to explore these enhancements. The current project provides a solid starting point for evaluating local LLM capabilities. With further improvements, it can become a comprehensive framework for testing and choosing the right model for on-premises NLP tasks. Happy benchmarking!