Local LLM Performance Analysis (Option 1: On-Premises LLM Setup)

Introduction

This report presents an analysis of deploying and benchmarking local Large Language Models (LLMs) on-premises (Option 1). The goal is to evaluate whether small, self-hosted models can meet our application needs in terms of performance and factual accuracy. We focus on three lightweight models, **Llama 3.2 (3B)**, **Qwen 2.5 (3B)**, and **Gemma 3 (4B)**, chosen for their manageable size (3-4 billion parameters), which allows them to run on consumer-grade GPUs. **Option 1 (Local LLM Setup)** was chosen for its potential benefits: privacy (data never leaves our infrastructure), cost savings on API calls, and control over customization. By running models locally, we avoid third-party dependencies and can fine-tune or integrate the models as needed. The trade-off is whether these smaller models can deliver acceptable accuracy and latency compared to hosted solutions (like GPT-4 or Claude). This analysis addresses that question through systematic benchmarks.

In summary, the assignment's objective is to gauge the viability of local LLMs for our use case. We define viability in terms of:

- **Throughput and latency:** Can the models generate responses quickly enough on local hardware?
- **Factuality:** Do the models produce factually correct answers (as measured against a ground truth dataset)?
- **Overall utility:** Considering both speed and accuracy, which model strikes the best balance for different hardware classes?

The following sections detail our benchmarking methodology, the hardware environments tested, results (with tables and graphs), and key insights. We conclude with recommendations on the best model choices for various hardware tiers and a brief comparison to hosted LLMs.

Methodology

Benchmark Tasks and Dataset

We evaluated the models on a set of factual question-answering tasks relevant to our product (Shram.io). A custom dataset of 10 questions was used, drawn from the file `shram_ai_dataset.json`. Each question pertains to factual details about Shram's mission, features, or impact. For example, one question asks: *"What problem does the original Shram.io product aim to solve and how does it recognize workers' contributions?"* For each question, a reference answer was available (ground truth) based on official documentation. This provided a basis to measure factual accuracy.

To simulate a realistic usage scenario, we employed a consistent prompt design that instructed the model to produce a concise, explanatory answer. The prompt template was:

```
"Write a brief story about Shram's mission and its impact on work using the
provided context."
```

This was followed by relevant context information for the question. In practice, for each question, the model received a short context snippet (e.g. a few sentences about Shram.io's features) and was asked to incorporate it into the answer. This approach tests the model's ability to use provided information accurately, analogous to how we might later add a vector database to supply context. All models were given identical prompts and context for fairness.

Each model answered all 10 questions. We recorded their outputs and compared them to the reference answers for content overlap. Additionally, we performed a separate throughput test (described below) to measure raw performance in terms of text generation speed.

Metrics

We assessed the models on several metrics:

- **Factuality Score (F1):** We computed a token-level F1 score for each model's answer against the reference answer. This score balances precision and recall of factual content. In essence, it measures how many key facts overlapped between the model's response and the ground truth. An F1 of 1.0 would indicate a perfectly factual answer containing all the expected information with no extraneous content. We averaged the F1 across the 10 questions to get an overall factuality score per model (per setting). This provides an objective measure of factual accuracy for each model.

- **Throughput (tokens/second):** To evaluate performance, we measured how fast each model can generate text. We conducted a standardized generation test for each model, using a fixed-length prompt and allowing it to produce a set number of tokens. The total tokens generated divided by the time taken gives the throughput in tokens per second. We also recorded **latency per query** (how many seconds on average to produce one answer) as a human-centric measure. A higher tokens/sec and lower latency indicate a more efficient model on the given hardware.

- **Diversity Index:** (Optional) We computed a diversity index to gauge the variability of the model's outputs. This index was defined as the proportion of unique tokens (or n-grams) in the model's generated text out of the total tokens. The aim was to see how creative or varied the outputs are, especially under different sampling settings. A higher diversity index means the model isn't simply repeating itself and is introducing varied vocabulary/phrasing. While not a primary metric for this factual Q&A task, it provides insight into the effect of sampling temperature on output variety.

- **Composite Score:** To combine accuracy and efficiency into a single figure of merit, we defined a composite score. We first normalized the factuality and efficiency metrics for all model runs onto a 0-1 scale (with 1 being the best performer in that metric across models). We then used a weighted sum:
  $$\text{Composite Score} = 0.6 \times (\text{Normalized Factuality}) + 0.4 \times (\text{Normalized Throughput})~,$$
  giving factual accuracy a 60% weight and throughput 40%. This weighting reflects that, for our use case, factual correctness is slightly more important than raw speed. The composite score helps identify the best overall model configuration per hardware by balancing the two objectives.

Variable Parameters

We experimented with one key generation parameter: the **temperature** of the sampling method. We tested each model at three temperature settings: **0.0** (greedy deterministic decoding), **0.7** (moderate sampling for some creativity), and **1.0** (high sampling for maximum creativity). Temperature is known to affect output: a low temperature yields more deterministic and focused responses (less randomness), while a high temperature increases diversity but can introduce more factual errors. By comparing these settings, we aimed to observe the trade-off between factual accuracy and creativity for each model, as well as any impact on generation speed. Each model thus had three runs (one per temperature) on each machine, resulting in 9 sets of results (3 models × 3 temperatures) per hardware environment.

All other generation parameters (such as maximum tokens, etc.) were kept constant across models. We ensured that the prompt format and context given were identical for every run, to attribute differences solely to the model and temperature.

Hardware Environments

We ran the benchmarks across 10 different machines to account for a range of hardware conditions. Each machine had a unique GPU and system configuration, from low-end to high-end, to see how the models perform on each. All machines ran Windows 10 or 11 and had recent multi-core CPUs (AMD Ryzen or Intel Core series) with at least 16 GB of RAM. Table 1 summarizes the GPU hardware for each test environment (sorted roughly from lower-end to higher-end):

Machine ID	GPU Model	GPU Memory
A-DIHH-VAASI	NVIDIA GeForce GTX 1650	4 GB
BEAST_BOY	NVIDIA GeForce RTX 3050 (Laptop)	4 GB
JEETH	NVIDIA GeForce RTX 2060 SUPER	8 GB
DESKTOP-SLV4UHA	NVIDIA GeForce RTX 2070 SUPER	8 GB
NARENKARTHICK	NVIDIA GeForce RTX 2070 SUPER	8 GB
VIBHEESH-PC	NVIDIA GeForce RTX 4060	8 GB

Machine ID	GPU Model	GPU Memory
KAUSHIK	NVIDIA GeForce RTX 3060 (Laptop)	8 GB
DESKTOP-1F6C84E	NVIDIA GeForce RTX 2060 (12GB)	12 GB
DESKTOP-9B492D4	NVIDIA GeForce RTX 4070 (Desktop)	12 GB
YUGENDRAN	NVIDIA GeForce RTX 4070 Ti	16 GB

Table 1: Hardware specifications of the 10 test machines. The GPUs range from an older GTX 1650 (4 GB VRAM) up to newer RTX 4070 Ti-class cards (with 12-16 GB VRAM). All systems used local GPU inference (with CPU offloading as needed when GPU memory was insufficient).

Each model was loaded onto each machine and run through the benchmarks. Notably, the variety in GPU memory and compute capability had a visible impact on performance. Machines with only 4 GB VRAM (e.g. A-DIHH-VAASI with a GTX 1650) struggled with the 4B model (Gemma), causing it to offload data to CPU and significantly slow down generation. In contrast, the high-end GPUs (12-16 GB, e.g. the RTX 4070/4070 Ti) easily handled all models entirely in GPU memory, achieving much higher throughput. We will highlight these differences in the results section. All machines were able to run the 3B models in GPU memory without issue; the 4B model was at the edge of feasibility on the 4 GB cards.

Apart from GPU differences, CPU differences were present (Intel vs. AMD, varying clock speeds), but these did not noticeably affect throughput in our tests since the decoding process was largely GPU-bound (except when GPU memory shortfall forced partial CPU use). The consistency of OS (Windows) and software (the same backend library for running LLMs) helped ensure comparability across environments.

Results

We present the benchmarking results in three parts: model throughput, factual accuracy, and the combined composite scores. For clarity, we averaged certain metrics across all machines to illustrate overall trends and include representative graphs. Where relevant, we also note specific per-hardware observations.

Throughput by Model and Temperature

Figure 1 (throughput graph, `throughput.png`) shows the text generation speed of each model at different sampling temperatures. The y-axis indicates each model's throughput in tokens per second (higher is better). Several observations can be made:

- **Smaller models run significantly faster.** The 3B-parameter models (Llama 3.2 and Qwen 2.5) achieved higher throughput than the 4B model (Gemma 3) on average. For instance, on a mid-range GPU (RTX 2060 Super), Llama 3B generated about 66 tokens per second, Qwen 3B about 62 tokens/sec, and Gemma 4B around 55 tokens/sec. On a high-end GPU (RTX 4070 Ti), Llama 3B exceeded 150 tokens/sec, Qwen 3B reached around 140, and Gemma 4B about 120 tokens/sec. The smaller models have fewer parameters to process per token, resulting in faster inference. Llama 3B proved to be the fastest model overall, slightly edging out Qwen 3B in tokens/sec on most hardware.

- **Temperature had minimal impact on speed.** The sampling temperature setting did not significantly affect throughput for any model. The lines for each model in Figure 1 are nearly flat across temperatures 0.0, 0.7, and 1.0, indicating only negligible variation in generation speed. This is expected because changing the randomness of token selection (via temperature) does not change the underlying computational cost of generating each token. There were tiny fluctuations (for example, Qwen 3B showed a very slight slowdown at temperature 0.7 on one machine, likely due to non-deterministic control flow), but these are within the margin of error. In summary, we can choose the temperature based on the desired creativity/factuality balance without worrying about any throughput penalty.

- **Hardware differences dominate speed outcomes.** (The values plotted are averaged across all 10 machines; the variation across hardware is indicated by error bars in the figure.) Low-end GPUs had much lower throughput than high-end GPUs. For example, on the weakest GPU (GTX 1650 with 4 GB), Llama 3B and Qwen 3B only managed on the order of 20-25 tokens/sec, and Gemma 4B fell below 10 tokens/sec when it had to offload to CPU. In contrast, the best GPU (RTX 4070 Ti) processed well over 100 tokens/sec for all models, an order-of-magnitude difference. This underscores that **GPU memory and compute capabilities are critical** for performance: a high-end card can deliver near-instant responses, whereas a low-end card may introduce noticeable delays (e.g. 10-15 seconds for an answer with Gemma 4B on a 4 GB GPU).

In practical terms, on a typical mid-level GPU (8 GB VRAM, e.g. RTX 2070 Super), Llama 3B took about 3 seconds on average to generate a full answer (~200 tokens) for a question, while on the RTX 4070 Ti it could do so in under 1.5 seconds. On the slowest GPU (GTX 1650), the same answer might take 10-15 seconds. These differences could influence real-time usability: high-end hardware can deliver near-instant answers, whereas very low-end hardware might make users wait.

Factual Accuracy by Model and Temperature

Figure 2 (factual accuracy graph, `factuality.png`) illustrates the average factuality score (F1 score) of each model's answers at each temperature. Higher values indicate more factual (ground-truth-aligned) responses. We observed the following:

- **Llama 3.2 (3B) produced the most factual answers.** It achieved the highest F1 score at all temperature settings. At $T=0$ (deterministic greedy decoding), Llama's average F1 was around 0.65 (meaning it included about 65% of the key facts from the reference answers on average). In comparison, Qwen 2.5 (3B) scored about 0.60, and Gemma 3 (4B) scored around 0.56. This suggests Llama 3B had a better grasp of the factual content or was better able to utilize the provided context to produce correct answers. We attribute this to Llama's strong pre-training (Meta's Llama family is known for solid knowledge even at smaller scales) and possibly to an improved fine-tuning or instruction following in version 3.2.

- **Qwen 2.5 (3B) was a close second in factuality.** Qwen's factual accuracy was only marginally behind Llama's in most cases. On some questions, Qwen matched Llama's correctness; on others it missed a detail that Llama captured, leading to a slightly lower average. Gemma 3 (despite having 4B parameters) lagged behind both 3B models in factual accuracy. Gemma's best-case F1 (at $T=0$) was about 0.57, indicating it often missed nearly half of the factual points. Its larger parameter count did not translate into better factual performance in our domain-specific test, possibly due to differences in its training focus or architecture.

- **Higher temperature generally reduced factual accuracy.** All models showed a decline in F1 as the sampling temperature increased. At $T=1.0$ (high creativity mode), Llama's F1 dropped to ~0.62, Qwen's plummeted to ~0.52, and Gemma's fell to ~0.54. This is expected: introducing more randomness can cause the model to stray from the facts or add extraneous content. A model at high temperature might "improvise" more, sometimes introducing inaccuracies or omitting details. Interestingly, Qwen 3B exhibited a slight *increase* in factuality when going from $T=0$ to $T=0.7$ (rising from 0.60 to about 0.61 F1) before dropping at $T=1.0$. This suggests that a small amount of randomness actually helped Qwen avoid deterministic mistakes or phrased certain facts better, but too much randomness was detrimental. In contrast, Llama and Gemma steadily declined as temperature rose, implying their $T=0$ outputs were already close to optimal for factuality, and any added creativity only introduced more error.

Overall, these F1 scores - mostly in the 0.55 to 0.65 range - reflect the challenge for small models to achieve perfect factual accuracy. Even with relevant context provided, the models did not capture every detail from the reference answers. For example, for the question about recognizing workers' contributions, a perfect answer would mention mechanisms like awarding XP points *and* badges (as the reference did). In our runs, Llama 3B did mention XP and badges, whereas Qwen 3B mentioned XP but missed badges in one of its answers, and Gemma 4B spoke only generally about "recognition" without specifics. Such omissions and generalizations explain the less-than-ideal F1 scores. None of the models achieved near-perfect factuality on this dataset, underscoring that **local 3-4B models have limitations in accurately conveying all facts**, even when given context. In a production scenario, we might need to fine-tune these models on our knowledge base or use retrieval augmentation to boost their factual accuracy (see the Discussion section on this).

Composite Score Comparison

The composite score combines each model's accuracy and speed into a single metric (with 60% weight on factuality and 40% on throughput). **Figure 3** (composite score graph, `composite.png`) compares the three models on this combined metric. For each model, we show the score at its optimal temperature setting (the temperature that yielded the highest composite score for that model).

Key findings from the composite comparison:

- **Llama 3.2 (3B) is the top performer overall.** It achieved the highest composite score among the models. This is primarily because Llama had the best factual accuracy without sacrificing much speed. In fact, on 7 out of the 10 test machines, Llama (typically at a moderate temperature like 0.7) delivered the highest composite score of any model configuration. For example, on one system with a 12 GB GPU, Llama 3B at $T=0.7$ nearly maxed out the composite scale (score ≈ 0.99 after normalization, meaning it was very close to the best in both accuracy and speed on that machine).

- **Qwen 2.5 (3B) is a close second, excelling on constrained hardware.** Qwen also shows a strong composite performance, just slightly behind Llama overall. On a few machines (3 of 10, particularly those with more limited hardware), Qwen actually achieved the highest composite score. Notably, on the lowest-end GPU (the 4 GB GTX 1650), Qwen at $T=0$ outscored Llama because Qwen ran significantly faster on that constrained device. In that scenario, Qwen's throughput advantage outweighed Llama's small accuracy edge. Similarly, on a mid-tier laptop GPU (RTX 3050 with 4 GB), Qwen's efficiency gave it a composite win. However, in environments where hardware was not a bottleneck, Llama's superior accuracy kept it in front.

- **Gemma 3 (4B) lags behind on the composite metric.** Gemma did not come out on top in composite score on any of the test machines. Its slower speed and slightly lower factuality combined to place it third. Even though Gemma's raw throughput on high-end hardware wasn't dramatically lower than the 3B models, its factual accuracy was consistently lower, and the composite metric (with heavier weight on accuracy) penalized that. Additionally, on lower-end GPUs, Gemma's performance suffered disproportionately (to the point of being impractically slow on a 4 GB VRAM device), which further hurt its normalized efficiency score. In short, Gemma's extra parameters did not yield a benefit for our tasks, so it ends up as the least favorable in the balance of speed and accuracy.

In summary, if we have to choose a single model and configuration that gives the best mix of speed and accuracy for local deployment, **Llama 3.2 (3B)** with a moderate temperature (around 0.7) would be the top choice in most cases. Qwen 2.5 (3B) is a close runner-up and might even be preferable for very resource-constrained GPUs due to its slightly better speed and memory handling. Gemma 3 (4B), while larger in size, did not translate that into better quality in our tests and is less efficient to run on local hardware.

Diversity Index

We also examined the diversity of the outputs, mainly to verify the effect of temperature on how varied the model responses were. The results were as expected:

At **temperature 0** (greedy decoding), each model's answers had very similar phrasing from run to run. The diversity index, measured as the fraction of unique words (unigrams) in the generated answers, was relatively low. For example, Llama 3B had a diversity index around 0.50 at $T=0$, meaning about half of the words in its answers were unique (the rest were repeats or common terms across answers). This indicates a lot of reuse of phrasing when no randomness is introduced.

At **temperature 1.0**, the diversity index rose substantially (to around 0.70-0.75 for these models). The models used a richer vocabulary and more varied sentence structures under high randomness. Both Qwen 3B and Llama 3B showed increases in lexical diversity with higher temperature, and Gemma 4B similarly produced more varied wording at $T=1.0$. However, this increased diversity often came at the cost of factual accuracy, as noted earlier. We observed that with $T=1.0$ the models sometimes included superfluous or off-script details in their "story" answers, essentially, the models were being more creative but occasionally straying from the facts.

While a higher diversity (more creative expression) is desirable in open-ended storytelling, in our factual Q&A context a too-diverse response often meant the model was introducing irrelevant or incorrect details. Thus, we treat diversity as a secondary metric. The main takeaway is that using a moderate temperature like 0.7 allows some variability in expression without a large hit to factuality, whereas pushing temperature to 1.0 greatly increases diversity but can reduce accuracy. In a real application, we would likely keep the temperature on the lower side for factual queries, to ensure the model remains focused and accurate.

Discussion

The results above provide several insights into running local LLMs on different GPUs and the trade-offs between performance and factual accuracy. Here we discuss these insights in more detail and consider their implications for deploying a local LLM-powered assistant.

Impact of GPU Hardware on Results

The range of GPUs tested, from a GTX 1650 up to an RTX 4070 Ti, demonstrated a clear stratification in model performance. High-end GPUs not only generated text much faster, but also avoided memory bottlenecks that can occur with limited VRAM. On the strongest GPU (RTX 4070 Ti with 16 GB), all models ran entirely in GPU memory and achieved very high throughput (well over 100 tokens/sec), meaning responses of a few hundred tokens could be produced in under 2 seconds. On mid-tier GPUs (e.g. RTX 2060/2070 Super or RTX 3060, generally 6-8 GB VRAM), the models still ran comfortably, just at proportionally lower speeds (roughly 20-70 tokens/sec depending on the model). Latency on these mid-tier machines was still reasonable for interactive use, typically a few seconds or less for an answer.

However, on the low-end 4 GB cards, we saw a dramatic slowdown, especially for the largest model (Gemma 4B). In one case, Gemma 4B's generation took around 15 seconds for an answer that higher GPUs did in 1-3 seconds, which is borderline unacceptable for a smooth user experience. The 3B models fared better on 4 GB: they could just fit with quantization, avoiding excessive CPU offloading. Qwen 3B in particular was relatively snappy (~20-25 tokens/sec on the 4 GB GPUs), likely due to optimizations or a slightly smaller memory footprint. Llama 3B was a bit slower on 4 GB (around 20 tokens/sec) but still far faster than Gemma 4B (~9-10 tokens/sec in the worst case). This indicates that **for low-end GPUs, model selection is crucial**, a slightly smaller or more efficient model can make the difference between a usable system and an unworkably slow one.

It's important to note that the *quality* of the outputs (factual accuracy) did not vary significantly with hardware; it's primarily a function of the model and prompt, not the GPU. In our tests, a Llama 3B running on a slow GPU produced essentially the same answer as it did on a fast GPU - it just took longer. All hardware produced the same content given the same model and input. The only indirect effect is that if hardware limitations forced us to use a smaller model or a shorter context, that could affect accuracy. But as long as the same model and inputs are used, a slower machine only impacts waiting time, not the answer's content. In summary, **the capabilities of the model are independent of hardware, but the user experience (speed of response) is highly dependent on hardware**.

Performance vs. Factuality Trade-offs

There is an inherent trade-off between a model's complexity (which affects both its knowledge and its speed) and its output factuality. One might assume a larger model will be more accurate, but our results show that's not always true at this scale. The 4B model (Gemma) did not outperform the 3B models in factuality, in fact, Llama 3B was more accurate. This suggests that model architecture and training data can matter more than sheer size when comparing 3B vs 4B parameters. Gemma's extra parameters didn't give it an edge on our domain-specific questions, but they did make it slower. In this case, **a well-tuned 3B model gave better accuracy per unit of computation than a 4B model**. It's a reminder that beyond a certain point, small models can't rely on scale alone; quality of training is key.

When adjusting the decoding temperature, we observed a different kind of trade-off: **factuality versus creativity.** At temperature 0 (fully deterministic), the models tended to give the most on-point and factually precise answers, however, those answers could be terse or formulaic. As we increased the temperature, the models produced more "story-like" and fluid answers, often more engaging to read, but at the risk of drifting from the facts. For example, at $T=1.0$ one model embellished an answer with a fictitious detail about "monthly ceremonies to celebrate top contributors" - something not present in our context data. This made the answer more colorful but less accurate. In a user-facing application, such hallucinated details can undermine trust in the system. Thus, we see that **injecting creativity (via higher temperature) comes at the cost of reliability** in factual tasks.

The good news is that this factuality-vs-creativity trade-off is adjustable without affecting performance: changing the temperature did not slow the model down. We can fine-tune the sampling to an optimal point (e.g., around 0.2-0.7) that yields answers which are sufficiently accurate yet not overly robotic, all while maintaining the same throughput. In practice, we would likely keep the temperature fairly low for any mission-critical factual queries to minimize the chance of hallucinations, only increasing it for tasks where creativity is explicitly desired.

Most Efficient Model and Why

Taking both speed and accuracy into account, **Llama 3.2 (3B)** stands out as the most efficient model in our tests. By “efficient,” we mean it achieves high accuracy per unit of computational effort. Llama’s strong factual performance means we get more correct information for the same number of tokens generated, and it ran quickly across all our hardware. In the composite score analysis, Llama consistently led because it didn’t force a hard compromise between speed and accuracy, it was good on both fronts.

Qwen 2.5 (3B) was not far behind. In fact, on the weakest devices, Qwen might be considered more efficient since it maintained speed where others slowed down. Qwen’s architecture or optimizations seem to give it an edge in low-memory environments. However, since we weigh accuracy a bit more heavily for our needs, Llama’s extra few percentage points of factuality made a difference, especially on hardware where speed was sufficient. If our use case put more weight on throughput (or if we had extremely limited hardware), Qwen could be viewed as equally or more efficient.

Gemma 3 (4B) was the least efficient by our measures. It required more computing work but did not provide a proportionate gain in answer quality, in fact, it scored lower in accuracy than the 3B models. This means a lot of its computation was essentially “wasted” from our perspective. It’s possible that Gemma has other strengths (for example, it might handle certain types of reasoning tasks better, or its larger size could help with longer context handling), but for our focused Q&A and short-context task, it wasn’t paying off. In terms of efficiency, **Gemma was doing more but achieving less** compared to Llama or Qwen.

Suitability of Local Models for Real-World Applications

Our findings indicate that these small local models **can** be suitable for certain real-world applications, with some caveats. On the positive side:

- **Real-time performance:** They are capable of running in real-time (or near real-time) on consumer hardware. Even a mid-range gaming laptop GPU could handle these

3B models with only a ~2-3 second delay for an answer, which is acceptable for many interactive applications or chat assistants. On a high-end desktop GPU, the responses are practically instantaneous for the user.

- **Coherent and relevant answers:** The models produced coherent, contextually relevant answers to our questions. In many cases, they effectively incorporated the provided context. The fact that Llama and Qwen achieved around 0.6 F1 on average without fine-tuning is encouraging, they clearly have some useful knowledge and language ability even at small scale. The answers were also phrased in a narrative, user-friendly way (as we requested), which could make them more engaging for end-users compared to a dry factual list.
- **Cost and privacy benefits:** The cost and privacy advantages of local deployment are significant. Once the models are running, each query incurs no external API cost, regardless of how many queries we handle. If our application expects heavy usage, this could save a lot of money compared to paying an API provider per call. Moreover, since all data processing stays on our machines, we avoid any privacy or compliance issues with sending sensitive information to third-party servers. This was a major motivation for exploring the on-premises option, and the tests show that it's feasible to achieve with these smaller models.

However, there are important limitations to acknowledge:

- **Factual accuracy is moderate:** As-is, the models' factual accuracy might not be high enough for a mission-critical or customer-facing application without additional safeguards. An average F1 of ~0.6 means the models often produce incomplete answers and occasionally incorrect details. For an internal knowledge base assistant, this could lead to user frustration or mistrust if the assistant fails to mention key points or introduces mistakes. In contrast, state-of-the-art large models (like GPT-4) would likely score much higher on the same questions. Bridging this gap would require techniques like fine-tuning on our data or using retrieval augmentation to provide the models with more information (see next subsection).
- **Tendency to hallucinate or go off-topic:** The small models can still hallucinate facts or stray from the context, especially if the prompt or temperature allows for creativity. We observed instances of this at higher temperature settings. In a real application, we would need to implement guardrails, for example, instructing the model to only use provided information, or having a verification step that checks the answer against source documents. Smaller models have even less world knowledge than large ones, so if they don't know something, they are more likely to make something up to fill the gap.
- **Maintenance and scaling considerations:** Running models locally means we have to maintain the infrastructure. If the usage grows or if we need high availability, we might need to set up multiple GPU servers or a load balancing system. This is a

manageable engineering task, but it’s additional effort compared to simply calling an API where the provider handles scaling. We also have to stay on top of updates, for instance, if a new version of an open-source model comes out that has better performance, we’d manually evaluate and deploy it. That said, given our scale and privacy requirements, these overheads are acceptable, but they are a factor to consider.

In summary, **local 3B-4B LLMs are quite viable for our needs, but they currently have a lower ceiling on accuracy and require some effort to ensure reliability**. For prototyping and even for production with modest demands, they can work well, especially when we control the domain and can supply context. To make them truly shine in a real-world setting, we should invest in improving their factual accuracy (below we discuss one major enhancement for that).

Benefits of Adding Vector Retrieval

One clear path to improve the factual accuracy of these local models is to integrate a **vector database retrieval** system into our pipeline. The idea is to use an embedding-based search to fetch relevant context documents from our knowledge base and provide those to the model along with the user’s query. In our benchmark, we manually provided a short context snippet for each question to simulate this, but a robust retrieval system would do it dynamically and with more information.

The anticipated benefits of adding a vector retrieval component are significant:

- **Higher factual accuracy:** If the model is given a detailed relevant document (or excerpt) from our knowledge base when answering a question, it can pull factual details directly from that text. This should greatly increase the likelihood of correct and complete answers. We would expect F1 scores to move much closer to 1.0 for questions where the information exists in the documentation. Essentially, retrieval can compensate for the model’s limited parametric knowledge by supplementing it with explicit knowledge.

- **Reduced hallucination:** When the model has concrete source text to reference, it’s less likely to fabricate facts. The model will tend to stay “grounded” in the provided material. We can further encourage this by prompt instructions (e.g., *“use the following document to answer…”*). This anchoring effect makes the system more trustworthy. The model’s role becomes more about summarizing or rephrasing the retrieved information rather than guessing.

- **Strong performance from smaller models:** With retrieval augmentation, even a small model can perform tasks requiring a lot of knowledge, because that knowledge is fetched on the fly. This means we might not need to jump to a larger 7B or 13B model to get better results; our 3B model could suffice when backed by the relevant data. This keeps our computational footprint and costs low while still delivering good answers. In other words, retrieval allows **the knowledge base to do the heavy lifting, letting the model focus on language and coherence**.

- **Easy domain updates:** We can update the vector database with new or corrected information at any time, and the model will immediately use it in responses. This is crucial for a dynamic domain like ours (for example, if Shram.io's features change or new data comes in). We would not have to retrain the model to teach it new facts; we just update our documents. This makes the system more maintainable and ensures up-to-date accuracy.

In short, adding a retrieval mechanism is the logical next step to make our local LLM setup far more powerful and reliable. Our experiments already demonstrated that the models can incorporate provided context reasonably well; with a dedicated retrieval pipeline feeding them rich context, their performance on factual queries should improve dramatically. We will likely need to implement this and adjust our prompting (e.g. explicitly telling the model to refer to the provided context, and perhaps even asking it to output answers with citations to the source). This hybrid approach, small LLM + retrieval, is a proven strategy to get the best of both worlds: the fluency of an LLM and the accuracy of a database.

Conclusion and Recommendations

In this assignment, we benchmarked three open-source lightweight LLMs in a local deployment scenario. We analyzed their speed and factual accuracy across a spectrum of hardware. Our findings highlight that a carefully chosen 3B-parameter model can serve as the foundation for a local AI assistant, provided the use case tolerates slightly lower accuracy than cutting-edge large models. Below are our actionable recommendations and final thoughts:

Best Model for Low-, Mid-, and High-End GPUs

Based on our multi-machine tests, we recommend different models depending on the available hardware:

- **Low-end GPUs (4 GB VRAM, older cards):** Use **Qwen 2.5 (3B)**. Qwen ran more efficiently under tight memory constraints, maintaining decent speed and accuracy where others slowed down. It appears to handle low VRAM conditions better, likely due to optimizations in its implementation. Llama 3B, while a bit more accurate, was slower on such hardware. If only a very low-end GPU (or CPU-only inference) is available, one might even consider a model smaller than 3B, but among our tested models Qwen 3B stood out for low-end scenarios.

- **Mid-range GPUs (6-8 GB VRAM, e.g. RTX 2060/2070 or RTX 3060):** Use **Llama 3.2 (3B)**. On these mid-tier cards, Llama ran well and delivered the highest factual accuracy. It strikes a good balance, its generation speed on an 8 GB GPU is still acceptable (only slightly slower than Qwen in some cases), and the boost in answer quality is worthwhile. Qwen 3B is a close second if for some reason Llama isn't available or if one absolutely needs that small extra bit of speed. We do not recommend using Gemma 4B on mid-range GPUs, as it is heavier on resources and did not provide an accuracy advantage in our tests.

- **High-end GPUs (≥ 10 GB VRAM, modern high-performance cards): Llama 3.2 (3B)** is again the top pick. With ample GPU memory and compute, Llama 3B was extremely fast and maintained the best factual accuracy. On high-end hardware, throughput differences between models become negligible (all are very fast), so maximizing accuracy is the priority, hence Llama's edge in factuality makes it the winner. Qwen 3B also runs very fast on a 12-16 GB GPU, but since Llama is equally fast there and a bit more accurate, Llama is preferable. Gemma 4B, while it can run fine on these GPUs, simply didn't offer a benefit over the 3B models in our evaluations.

Best Overall Model for Our Use Case

Considering all factors, if we have to choose one model to deploy as a local assistant for Shram.io, **Llama 3.2 (3B)** emerges as the best overall choice. It has demonstrated the highest factual accuracy, robust performance across different hardware, and benefits from the extensive community support around the Llama family (which means we might find better quantization or fine-tuning techniques for it readily available).

That being said, we will keep **Qwen 2.5 (3B)** in mind as a very viable alternative or complement. Qwen was only slightly behind Llama in our specific tests and may have strengths not fully captured by our metrics. For example, some anecdotal observations suggest Qwen has strong reasoning or coding abilities for its size, which could be useful if our use case expands beyond straightforward Q&A. If we later introduce more creative tasks or complex multi-step queries, Qwen might shine. In any case, both of these 3B models have proven capable, and having a backup model is useful in case one has difficulties with certain queries (or for load balancing).

In summary, **Llama 3B is our primary recommendation** for a local LLM, with **Qwen 3B as a backup/secondary option**. Either can be the core of our on-premises AI system, with Llama having a slight edge in our factual domain.

Comparison to Hosted LLMs (GPT-4, Claude, etc.)

It is important to put our local LLM approach in context by comparing it with state-of-the-art hosted LLM services:

- **Accuracy and Capability:** Large models like OpenAI's GPT-4 or Anthropic's Claude have on the order of 100+ billion parameters and have been trained on vast amounts of data. They would likely answer the Shram.io questions with near-perfect accuracy and even provide more elaborate, well-structured explanations. In contrast, our 3B models sometimes miss details or make errors. For complex or high-stakes queries, GPT-4’s advanced reasoning and knowledge means it will significantly outperform a 3B local model. This gap in capability is the trade-off we accept for the benefits of a local model. We should be aware of these limitations, for certain types of queries (especially ones requiring extensive world knowledge or nuanced understanding), our local model might not suffice without retrieval help or fine-tuning, whereas GPT-4 would excel.

- **Latency:** Interestingly, our small local models on a good GPU can respond as fast as or faster than hosted models in many cases. GPT-4’s API often has a few seconds of overhead (network latency and the model’s own processing time), and if the output is lengthy, it streams gradually. Our local Llama or Qwen on a high-end GPU can start producing tokens immediately and finish a short answer in under 2 seconds. For longer responses, GPT-4 is also quite fast (it has more raw power), but the difference is not huge for moderate lengths. Overall, both approaches can provide near real-time interaction, but the local model has the advantage of **no reliance on internet connectivity** or external service uptime. If our application needs guaranteed availability and consistent response times, local deployment avoids any external bottleneck.

- **Cost:** Using GPT-4 or Claude via API incurs a cost per call (often based on token usage), which can add up quickly if we have many queries or long documents to analyze. Our local model, after the initial investment in hardware, can handle unlimited queries at essentially no additional cost. If we foresee heavy usage (for example, an assistant that employees use throughout the day, or a public-facing chatbot with many users), the cost savings of using local models will be substantial. We also avoid any sudden pricing changes or limits imposed by third-party providers.

- **Privacy:** With hosted models, any data we send (such as the context about our projects or user queries) goes to an external server. For sensitive business data, this can be a legal or trust concern. By keeping everything on-premises, we ensure that confidential information never leaves our controlled environment. This is a decisive advantage for applications dealing with proprietary or personal data, and one of the core reasons we explored the local LLM option from the start.

- **Maintenance:** Hosted LLMs are maintained, updated, and improved by their providers, which is convenient. Our local model setup will require us to handle updates and improvements ourselves. However, because the models are open-source, the community often releases fine-tuned versions or new techniques (for example, an optimized int8 quantization, or a fine-tune on a Q&A dataset) that we can adopt. We have the flexibility to customize the model (even fine-tune it on our specific data) which isn’t possible with closed APIs. This control is beneficial, but it means we shoulder the responsibility to keep the system up-to-date and performing well.

In conclusion, while our local LLM solution cannot yet match the raw power and accuracy of something like GPT-4, **it offers a strong value proposition for our needs**. We get fast, coherent answers at zero per-query cost and with full control over our data. With the planned enhancements (particularly retrieval augmentation), we expect to close much of the accuracy gap for our specific domain. For the majority of day-to-day questions about Shram.io, a locally running 3B model should suffice. We will remain mindful, though, that for some very complex questions or tasks, a larger model might still be needed - and we could then consider a hybrid approach (using the local model by default and falling back to an API for the hardest queries, if necessary).

Next Steps and Future Work

Going forward, we plan to implement the following steps to transition from this analysis into a deployed solution:

1. **Deploy Llama 3.2 (3B) as the default local model**, with Qwen 2.5 (3B) available as a backup. We will set up the serving infrastructure (ensuring the model is loaded with 4-bit quantization on our GPU server for efficiency).

2. **Use a moderate temperature setting** (around 0.5 to 0.7) for generation by default. This will keep answers mostly factual and focused, while still allowing a bit of natural variability in phrasing. We may adjust this setting per query if needed (e.g., lower for strictly factual queries, higher if a more creative response is acceptable).

3. **Integrate a vector database for context retrieval.** We will index our Shram.io documentation, FAQs, and other knowledge sources. For each user query, we'll retrieve the most relevant snippets and prepend them to the model's prompt. This should significantly boost the factual accuracy of the responses and reduce hallucinations. We will also craft the prompt to encourage the model to use that information (and possibly to cite it).

4. **Test the improved system against GPT-4 on a sample of queries** as a benchmark. This will give us a sense of the remaining gap in answer quality. If we find that for certain question types GPT-4's answers are dramatically better, we will consider a hybrid approach: for those particular cases (complex, rare queries), the system could call a hosted API. However, our goal is to handle the vast majority of queries locally, so we will work on fine-tuning and prompt engineering to minimize the need for fallbacks.

By following these steps, we expect to achieve an **efficient, cost-effective, and reasonably accurate** AI assistant powered entirely by local LLM technology. This validates the choice of Option 1 for our needs. It also sets the stage for further enhancements: in the future, we could explore fine-tuning the model on transcripts of our own Q&A sessions to further improve its alignment with our domain, or evaluate new open-source models in the 3B-7B range as they become available.

Overall, we are confident that a local Llama/Qwen 3B model augmented with retrieval is the right path forward for Shram.io's AI assistant, offering a good balance of privacy, performance, and accuracy for our on-premises deployment.
