COST-EFFICIENT COMMUNICATION BETWEEN ON-DEVICE AND CLOUD LANGUAGE MODELS

Avanika Narayan*[1], Dan Biderman*[1], Sabri Eyuboglu*[1], Avner May[2], Scott Linderman[1], James Zou[1], and Christopher Ré[1]

Stanford University [1] Together AI [2]
{avanikan,biderman,eyuboglu}@stanford.edu

ABSTRACT

We investigate an emerging setup in which a small, on-device language model (LM) with access to local data communicates with a frontier, cloud-hosted LM to solve real-world tasks involving financial, medical, and scientific reasoning over long documents. *Can a local-remote communication reduce cloud inference costs while preserving quality?* First, we consider a naïve communication protocol where the local and remote models simply chat back and forth. Because only the local model reads the full context, this protocol achieves a $30.4\times$ reduction in remote costs, but fails to recover the performance of the frontier model. We identify two key limitations of this protocol: the local model struggles to (1) follow the remote model's multi-step instructions and (2) reason over long contexts. Motivated by these observations, we study an extension of this protocol, coined MINION$\mathcal{S}$, in which the remote model decomposes the task into easier subtasks over shorter chunks of the document, that are executed in-parallel locally. MINION$\mathcal{S}$ reduces costs by $5.7\times$ on average while recovering 97.9% of the performance of the remote model alone. Our analysis reveals several key design choices that influence the trade-off between cost and performance in local-remote systems.

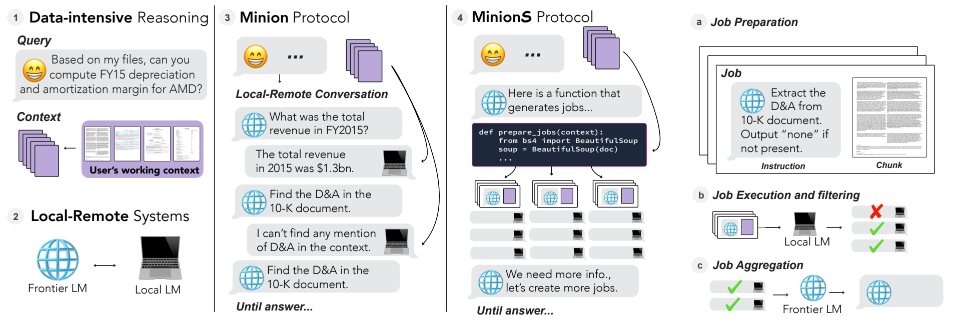
1 INTRODUCTION

Today's cloud-hosted frontier Language Models (LMs) can perform *data-intensive reasoning*: they can program across repositories and make decisions based on financial, legal, and medical documents. However, accessing these models is expensive: processing a standard million-token code repository with OpenAI's o1 API costs $> \$15$ per query. At the same time, smaller LMs (1-8B parameters) are rapidly improving and can now run on personal computers (Ollama, llama.cpp) and smartphones Mehta et al. (2024); Yi et al. (2024); Xu et al. (2024)). Yet, today, these small, on-device LMs are used mostly for simple tasks such as tone adjustment and text completion Gunter et al. (2024). They do not play a role in data-intensive reasoning tasks.

Inspired by the growing literature on multi-agent systems (Wang et al., 2024; Guo et al., 2024), in this work we ask: *how can a small LM on-device collaborate with a frontier LM in the cloud to reduce inference costs on data-intensive reasoning tasks?* In particular, we study the *communication protocols* that govern how the two LMs talk to each other. To mimic realistic use cases, we study tasks that involve varying levels of reasoning over large volumes of medical, financial, and academic data (Islam et al., 2023; Adams et al., 2024; Dasigi et al., 2021).

As our first attempt, we study a simple communication protocol: an unconstrained chat between the local and remote models. This protocol, coined MINION, reduces cloud costs by only "reading" the data locally, and communicating a compressed version of the context to the remote model. We show that while MINION achieves a $30.4\times$ reduction in remote model costs, it trails behind the remote-only baseline by 9.4 accuracy points on average (with an 8B model; see Section 4 for details). We identify two key limitations of small LMs that hinder MINION's performance (Section 4.1):

- *Small LMs struggle following multi-step instructions.* We find that splitting complex instructions into separate requests improves performance by 56%.


1 Data-intensive Reasoning
Query
Based on my files, can you compute FY15 depreciation and amortization margin for AMD?
Context
User's working context
2 Local-Remote Systems
Frontier LM
Local LM
3 Minion Protocol
Local-Remote Conversation
What was the total revenue in FY2015?
The total revenue in 2015 was $1.3bn.
Find the D&A in the 10-K document.
I can't find any mention of D&A in the context.
Find the D&A in the 10-K document.
Until answer...
4 MinionS Protocol
Here is a function that generates jobs...
We need more info., let's create more jobs.
Until answer...
a Job Preparation
Job
Extract the D&A from 10-K document. Output "none" if not present.
Instruction
Chunk
b Job Execution and filtering
Local LM
c Job Aggregation
Frontier LM

```
def prepare_jobs(context):
    from bs4 import BeautifulSoup
    soup = BeautifulSoup(doc)
    ...
```

Figure 1: **Local-Remote Systems**. MINION and MINION$\mathcal{S}$ protocols. **(Left)** Problem set-up: local and remote LM collaborate on a data-intensive reasoning task. **(Center)** MINION: A simple communication protocol in which the local and remote models have an "unconstrained" back and forth chat. **(Right)** MINION$\mathcal{S}$: an extension of MINION where the remote LM decomposes a query into many *jobs* that are processed in parallel by the local model. Each job is a single-step instruction over a chunk of the context.

- *Small LMs get confused by long contexts*. Increasing context length from $< 1K$ to $> 65K$ decreases performance by 13% on a simple extraction task.

Motivated by these limitations, we propose MINION$\mathcal{S}$, an extension of MINION where the remote LM decomposes the problem into a set of *single-step instructions* to be performed on smaller *chunks* of the document. Crucially, the remote model has to do this *without* reading the full document, which it achieves by *generating code* that is later executed locally where the document is. More precisely, MINION$\mathcal{S}$ involves a loop over three steps:

1. **Decompose**: Given a task, the remote model writes code that decomposes it into "bite-sized" *subtasks*.
2. **Execute**: The local LM then executes the subtasks in parallel and sends a filtered selection of the responses back to the remote model.
3. **Aggregate**: The remote model aggregates the local outputs and finalizes the solution or loops back to the Decompose step.

Averaged across tasks, MINION$\mathcal{S}$ with an 8B parameter local LM can recover 97.9% of the performance of remote-only systems at 18.0% of the cloud cost. With a 3B parameter local LM, MINION$\mathcal{S}$ achieves 93.4% of the performance of remote-only systems at 16.6% of the cloud cost.

We perform a detailed analysis of the design and hyperparameter space of MINION$\mathcal{S}$. Our analysis highlights several "knobs" that allow us to trade off cost for quality.

(a) Model choice *How does the size and family of the language models affect the cost and quality?* We show that MINION$\mathcal{S}$ would not have been feasible until mid-2024 and is now performant with the latest 3B-parameter models running locally.

(b) Scaling parallel workloads on-device. *How should we structure parallel workloads at the edge to maximize performance?* In Section 6.3, we study three different strategies for increasing the parallel workload on-device: (a) repeated sampling, (b) decomposition, and (c) context chunking. We show that all three can independently improve quality at the expense of increased remote cost.

(c) Sequential communication protocols. *Can multiple rounds of communication improve quality? At what cost?* We show that by increasing the number of sequential rounds of communication, we can pay more to improve quality.

To summarize, our main contributions are as follows:

- Propose a naive local-remote LM communication protocol that achieves $30.4\times$ efficiency over remote-only workloads while recovering 87% of performance.
- Propose MINION$\mathcal{S}$, an extension that overcomes the limitations we identify in MINION, achieving $5.7\times$ cost-reduction over remote-only workloads and recovering 97.9% of performance.

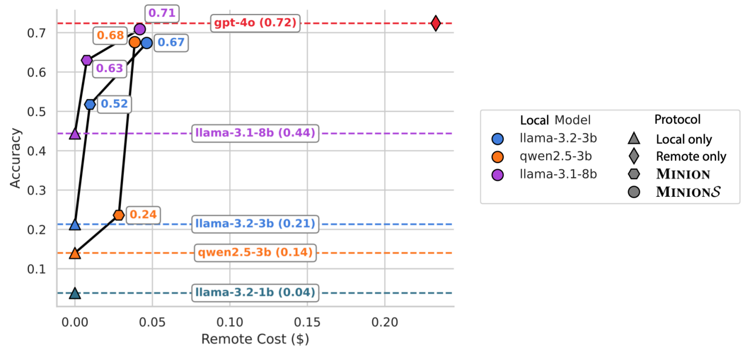


Figure 2: **Cost-Accuracy Tradeoff in Edge-Remote Systems.** Macro-average accuracy (y-axis) vs. cost (x-axis) across FINANCEBENCH (Islam et al., 2023), LONGHEALTH (Adams et al., 2024), and QASPER (Dasigi et al., 2021). Accuracy represents the fraction of correct predictions, while cost is the average USD per query based on GPT-4o rates (Jan 2025: \$2.50/1M input tokens, \$10.00/1M output tokens); see section 3. The table compares Naïve (section 4) and MINION$\mathcal{S}$ (section 5) protocols against local-only and remote-only baselines. Points, colored by local model, use GPT-4o as the remote model. Exact metrics in table 1.

- Conduct an in-depth analysis of MINION$\mathcal{S}$, exploring design choices to traverse the cost-accuracy trade-off.

2 RELATED WORK

See Appendix A for an extended discussion of related work.

We are inspired by a large body of work that studies how to combine multiple LMs and tools to improve quality and reduce cost of cloud workloads. These include multi-agent system (Guo et al., 2024; Wang et al., 2024), compound LM systems (Saad-Falcon et al., 2024; Khattab et al., 2023; Yuksekgonul et al., 2024), model-routing (Chen et al., 2024a; 2023), and retrieval-augmented generation (Lewis et al., 2020; Karpukhin et al., 2020; Lee et al., 2019). We differ from these works by studying the specific asymmetric cost model that arises from the local-remote setting.

The techniques used in MINION$\mathcal{S}$ build upon several ideas proposed in the literature, including LM orchestration and memory systems to support long-context reasoning (Packer et al., 2023; Jayalath et al., 2024; Russak et al., 2024; Shankar et al., 2024; Zhou et al., 2024), prompting and decomposition techniques for improved small LM quality (Arora et al., 2022; Patel et al., 2022; Wu et al., 2022), scaling local compute via test-time sampling and verification (Brown et al., 2024; Song et al., 2024; Hassid et al., 2024; Snell et al., 2024; Wu et al., 2024), and using code to facilitate reasoning (Arora et al., 2023; Li et al., 2023)

Some recent works have explored aspects of the local-remote setting. Several study how local-remote systems can limit leakage of private information to a cloud-hosted LM API siyan2024papillon, zhang2024cogenesis. In this work, we do not address privacy concerns, though these privacy techniques can be used in conjunction with MINION$\mathcal{S}$. Others have explored efficient routing patterns between local and remote computation for LM workloads, albeit without two models communicating in natural language or collaborating on a solution (Jin & Wu, 2024; Yang et al., 2024).

3 PRELIMINARIES

We study the tradeoff between the *quality* of a local-remote system and the cost of running it. We first outline the problem setup and then provide details on how we measure accuracy and cost.

Problem setup We study language tasks that involve a context $\mathbf{c}$ (*e.g.* a long document), a query $\mathbf{q}$ against that context, and a ground-truth answer $\mathbf{y}$ (see (1) in Figure 1).

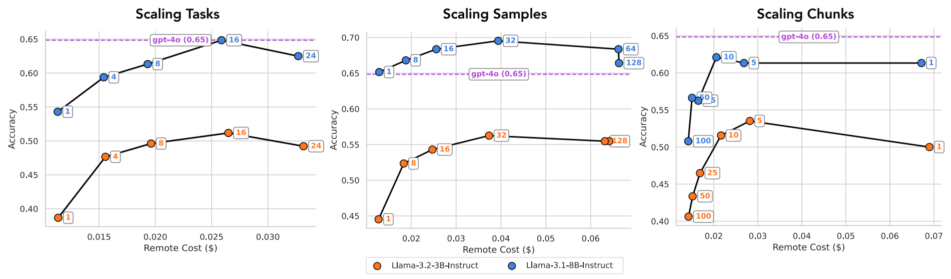


Figure 3: **Scaling parallel jobs on-device improves quality.** The x-axis represents tokens processed by the *remote model*, and the y-axis shows macro-average accuracy across LongHealth and QASPER. The cloud model is GPT-4o. Each plot varies a different Minion$\mathcal{S}$ hyperparameter affecting parallelism, with annotated values. **(Left)** Varying the number of unique *instructions*. **(Middle)** Varying the number of unique *samples*. **(Right)** Varying the chunking granularity in code $\mathbf{f}$. See section 5 for details.

Context (c): `Apple's FY24 10-K report`
Query (q): *What is the total revenue for the year?*
Answer (y): `US$394,328 million`

A *local-remote* system S (see (2) in Figure 1), consists of two language models that must collaborate to solve the task—a small LM (LocalLM) running on on-device, and a large frontier LM (RemoteLM) running in the cloud. S ingests a context and an associated query, and applies both models in conjunction to output a predicted answer: $\hat{\mathbf{y}} \sim \mathrm{S}(\mathbf{c}, \mathbf{q})$.

Measuring quality We evaluate the performance of S on a dataset $\mathcal{D} = \{(\mathbf{c}_i, \mathbf{q}_i, \mathbf{y}_i)\}_{i=1}^{N}$, via a scoring metric $s(\hat{\mathbf{y}}_i, \mathbf{y}_i)$. Here, we use $s(\cdot,\cdot)$ is binary (correct/incorrect) and we report *accuracy*. As baselines, we compare S to $\hat{\mathbf{y}}_{\text{remote}} \sim \mathrm{RemoteLM}(\mathbf{c}, \mathbf{q})$ and $\hat{\mathbf{y}}_{\text{local}} \sim \mathrm{LocalLM}(\mathbf{c}, \mathbf{q})$.

Measuring cost Monetary cost (in $USD) is our primary cost metric. We assume that RemoteLM calls incur a cost while LocalLM calls (for $\leq$ 8B parameter models) are free, ignoring the fixed cost of the hardware and marginal cost of energy consumption.

More concretely, the cost of calls to RemoteLM are proportional to a weighted sum of the number of prefill (*i.e.* input) tokens and decode (*i.e.* output) tokens:

$$C_{\text{remote}}(n_{\text{prefill}}, n_{\text{decode}}) \propto n_{\text{prefill}} + \alpha \cdot n_{\text{decode}}$$

Where α varies by provider ($\approx$1– 5) (Dubey et al., 2024; Anthropic, 2024). Decode tokens are more expensive since the decoding stage is IO-bound (Leviathan et al., 2023) and has lower GPU utilization[1].

In this work we do not focus on optimizing the latency of local-remote systems. However, in appendix C we show analytically that there are important regimes where the systems proposed (Minion, Minion$\mathcal{S}$) incur at most a $5\times$ increase in latency relative to performing the entire operation remotely. This is possible because these systems avoid the costly step of processing the entire document with the huge RemoteLM, and because they can make efficient use of the local hardware by batching (*e.g.* in Minion$\mathcal{S}$). We leave a detailed empirical study of the latency trade-offs of these local-remote systems for future work.

4 Minion: A naïve communication protocol

In this section, we describe Minion, a baseline local-remote communication protocol, which implements a simple free-form conversation between LocalLM and RemoteLM.

It begins with system prompts for both models informing them of the query $\mathbf{q}$ and that they will be collaborating with another model to answer it (see (3) in Figure 1). Crucially, the system prompt

[1]Generating each decode token requires loading the full model and KV cache into GPU SRAM.

for the LocalLM does include the full context $\mathbf{c}$ while the system prompt for the RemoteLM does not. After the system prompts, the models chat back and forth until the RemoteLM provides a final answer to the query. *See Appendix D.1 for a detailed description of the* MINION *protocol.*

4.1 ANALYSIS

We compare MINION to a baseline where RemoteLM is given the full context and the query. Excitingly, MINION reduces RemoteLM costs by $38.13\times$, $31.3\times$, and $20.9\times$ on FINANCEBENCH, LONGHEALTH and QASPER, respectively. Averaged across these datasets, it closes 87.0% of the quality gap between RemoteLM and LocalLM operating alone.

To close the gap, we analyze MINION conversations and find that in unconstrained chat, RemoteLM often gives LocalLM *complicated instructions* over *long contexts*. Appendix E.2 presents micro-experiments illustrating LocalLM's struggles with these instructions:

1. LocalLM **struggles to handle multi-part instructions.** Using GPT-4o, we generate instructions with varying numbers of sub-parts. We then show splitting sub-parts into separate requests leads to a 56 point performance improvement (see Figure 6).
2. LocalLM **struggles to reason across long contexts**. We show how increasing context length from $< 1K$ to $> 65K$ tokens can decrease performance by 13% on a simple extraction instruction (see Figure 6).

Put simply, these models are currently better equipped to answer simple queries on shorter contexts.

5 MINION$\mathcal{S}$: A SIMPLE, DECOMPOSITION-BASED COMMUNICATION PROTOCOL

Motivated by these observations, we introduce MINION$\mathcal{S}$, a simple extension of the naïve communication protocol discussed in section 4. MINION$\mathcal{S}$ uses a divide-and-conquer strategy where the RemoteLM decomposes the task into *simpler* jobs that can be run *in parallel* (see (4) in Figure 1).

5.1 PROTOCOL DESCRIPTION

MINION$\mathcal{S}$ protocol is a loop around three steps:

1. *Job preparation on remote.* RemoteLM writes code that generates a list of job specifications for LocalLM (see 4(a) in Figure 1).
2. *Job execution and filtering locally.* The job specifications are executed locally with the LocalLM and outputs are filtered (see 4(b) in Figure 1).
3. *Job aggregation on remote.* The remote model receives the filtered outputs and decides whether to output an answer or begin another iteration (see 4(c) in Figure 1)

Throughout we will continue with this example task:

Compute the 2015 depreciation and amortization margin for AMD (in percentage).

Step 1: Job preparation on remote. In this step, the RemoteLM generates a list of jobs that the LocalLM will run in parallel. A *job* is a specification of a subtask, which can be converted into a prompt and sent to the local model. More precisely, a job, $\mathbf{t}$, is a context-instruction pair $\mathbf{t}^{(i)} = (\tilde{\mathbf{q}}^{(i)}, \tilde{\mathbf{c}}^{(i)})$. We denote a list of jobs with $\mathbf{T} = [\mathbf{t}^{(1)}, \mathbf{t}^{(2)}, ...]$

Instruction ($\tilde{\mathbf{q}}^{(i)}$): Extract the total revenue for FY2015, abstain if not present. Try look for the income statement and make sure it is from 2015.
Context ($\tilde{\mathbf{c}}^{(i)}$): "Operating income for North America for the years ended..."

Crucially, the context $\tilde{\mathbf{c}}^{(i)}$ for a job need not include the entire context $\mathbf{c}$ of the full task. In principle, this allows us to chunk the context into more manageable pieces, which can be executed in parallel. *But how can the* RemoteLM *chunk the context without reading it?*

To avoid reading the entire context, we have the remote model program a function, $\mathbf{f}(\mathbf{c}, \mathbf{T})$, that accepts the full task context $\mathbf{c}$ and jobs from the last iteration $\hat{\mathbf{T}}$ and outputs a new list of jobs $\mathbf{T}$. Specifically, we prompt $\mathrm{RemoteLM}$ with the task query $\mathbf{q}$ and instruction prompt $\mathbf{p}_{\text{decompose}}$: $\mathbf{f}(\cdot, \cdot) \sim \mathrm{RemoteLM}(\mathbf{q}, \mathbf{p}_{\text{decompose}})$. Then, on-device, the function is executed with the context $\mathbf{c}$ as the argument producing a list of jobs $\mathbf{T} = \mathbf{f}(\mathbf{c}, \hat{\mathbf{T}})$.

This strategy, which builds on work using LMs to generate code for information extraction (Arora et al., 2023; Li et al., 2023), allows us to decouple the number of unique jobs from the number tokens generated by the cloud model. For example, the code below, which is an abbreviated version of a function that was generated by the cloud model, is < 15 lines but can generate hundreds of jobs.

```
@dataclass
class Job:
   instruction: str
   chunk: str
def f(ctx: str,lastjobs: List[Job])->List[Job]:
   jobs = []
   instructions = ["Extract the total revenue for...", "In the statement of cash flow..."]
   for chunk in chunk_on_pages(ctx):
      for instr in instructions:
         for _in range(5):
            jobs.append(Job(instr, chunk))
   return jobs
```

Additionally, by passing the previous iteration's jobs and responses $\mathbf{T}$ (`last_jobs` in the code snippet), the large model can create jobs which build on previous responses. For example, the cloud model in the second round might zoom in on a relevant chunk identified in the first round. For more examples of generated functions or prompts used to generate the code, see appendix F.

Step 2: Job execution and filtering on-device. In this step, we convert the jobs $\mathbf{T} = [\mathbf{t}^{(1)}, \mathbf{t}^{(2)}, ...]$ into prompts and execute them in parallel locally.

The jobs are fed in batch(es) to the $\mathrm{LocalLM}$ together with a system prompt $\mathbf{p}_{\text{worker}}$ that instructs the model to either abstain or return a JSON object $\mathbf{z}^{(i)}$ with fields `answer` and `explanation` to help verify its reasoning.

$$\mathbf{z}^{(i)} \sim \mathrm{LocalLM}(\mathbf{t}^{(i)}, \mathbf{p}_{\text{worker}}) \tag{1}$$

After the $\mathrm{LocalLM}$ has generated the results, we discard any $\mathbf{z}^{(i)}$ for which the model abstained. Intuitively, many instructions will be irrelevant to their paired chunks, allowing the $\mathrm{LocalLM}$ to abstain and avoid sending unnecessary information to the $\mathrm{RemoteLM}$. The surviving subtask-chunk pairs are aggregated to form the formatted string $\mathbf{w}$.

Step 3: Job aggregation on remote. $\mathrm{RemoteLM}$ receives $\mathbf{w}$ and a synthesis prompt $\mathbf{p}_{\text{synthesize}}$, instructing it to generate a JSON object $\mathbf{a}$ with a "decision" field for sufficiency and a "response" field for a (potential) final answer:

$$\hat{\mathbf{y}} \sim \mathrm{RemoteLM}(\mathbf{w}, \mathbf{p}_{\text{synthesize}}) \tag{2}$$

If the $\mathrm{RemoteLM}$ decides that more information is needed, the loop continues from Step 1.

There are several ways to maintain context across MINION$\mathcal{S}$ rounds. One simple approach is to keep the entire the conversation in context. However, this strategy incurs significant additional cost, even with prompt caching. We experiment with two alternatives: (1) *simple retries*, in which only the $\mathrm{RemoteLM}$'s advice is carried over between rounds and (2) *scratchpads*, in which the $\mathrm{RemoteLM}$ can record what it learned from the round before proceeding to the next.

5.2 PROTOCOL HYPER-PARAMETERS

MINION$\mathcal{S}$ has three hyper-parameters: choice of $\mathrm{RemoteLM}$ and $\mathrm{LocalLM}$ (model choice), job preparation strategy (scale of parallel workloads on-device), and looping strategy (sequential communication protocol).

Model choice. Different model sizes (*e.g.* 3B vs. 8B), families (*e.g.* QWEN2.5 vs. LLAMA), and generations (*e.g.* 3.1 vs. 3.2) can be used for both the $\mathrm{LocalLM}$ and the $\mathrm{RemoteLM}$.

Protocol	Local Model	Remote Model	Macro Avg. Acc.	Macro Avg. Cost	FinanceBench Acc.	FinanceBench Cost	LongHealth Acc.	LongHealth Cost	QASPER Acc.	QASPER Cost
Remote Only	—	GPT-4o	**0.724**	$0.233	**0.826**	$0.261	**0.748**	$0.301	<u>0.598</u>	$0.137
Local Only	Llama-8B	—	0.444	$0.000	0.326	$0.000	0.468	$0.000	0.538	$0.000
Local Only	Llama-1B	—	0.038	$0.000	0.000	$0.000	0.115	$0.000	0.000	$0.000
Local Only	Llama-3B	—	0.213	$0.000	0.130	$0.000	0.345	$0.000	0.164	$0.000
Local Only	Qwen-3b	—	0.140	$0.000	0.087	$0.000	0.177	$0.000	0.156	$0.000
Minion	Llama-8B	GPT-4o	0.630	$0.008	<u>0.804</u>	$0.007	0.635	$0.010	0.450	$0.007
Minion	Llama-3B	GPT-4o	0.518	$0.010	0.698	$0.010	0.482	$0.009	0.372	$0.011
Minion	Qwen-3b	GPT-4o	0.236	$0.028	0.217	$0.029	0.281	$0.021	0.210	$0.035
Minion$\mathcal{S}$	Llama-8B	GPT-4o	<u>0.709</u>	$0.042	<u>0.804</u>	$0.053	<u>0.740</u>	$0.054	0.582	$0.019
Minion$\mathcal{S}$	Llama-3B	GPT-4o	0.662	$0.052	0.726	$0.079	0.703	$0.057	0.558	$0.020
Minion$\mathcal{S}$	Qwen-3b	GPT-4o	0.676	$0.039	0.783	$0.059	0.645	$0.043	**0.600**	$0.015

Table 1: **Accuracy and cost of local-remote systems.** Evaluation of cost and accuracy on 3 datasets. The table compares two edge-remote protocols—Naïve (section 4) and Minion$\mathcal{S}$ (section 5)—against edge-only and remote-only baselines. We assess 3 local models and 1 remote model. Cost (USD) is the average per-query expense, based on GPT-4o rates (Jan 2025: $2.50M/input tokens, $10.00M/output tokens). Local model execution is assumed free (see section 3 for cost details).

Scale of parallel workload on-device. Minion$\mathcal{S}$ has three knobs for increasing the degree of task decomposition and thus, workload parallelization: (1) number of tasks per round (*i.e.* "Extract the ARR for Q1 of 2014"), (2) number of samples per tasks (*i.e.* any integer value > 1), and (3) number of chunks (*i.e.* chunk by page, paragraph, etc.). These parameters can be configured by $\mathrm{RemoteLM}$.

Sequential communication protocol. In practice, it is important to cap the number of loops for Minion$\mathcal{S}$. After this limit, the synthesis prompt is adjusted to produce a final answer. This maximum affects accuracy and cost, and context management strategy (simple retries vs. scratchpads) is another key hyperparameter.

We analyze these hyperparameters in Section 6.

6 Results

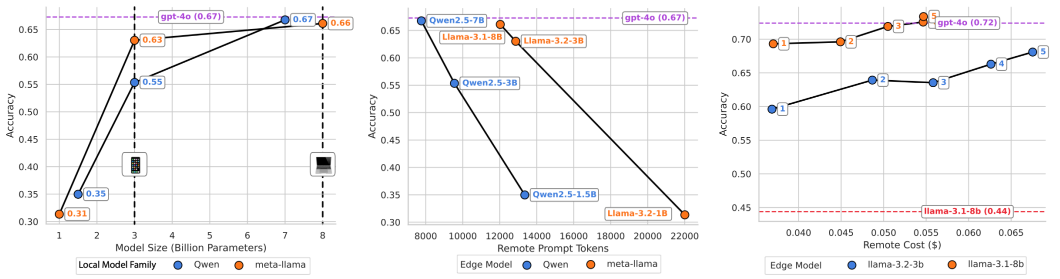


Figure 4: **Trade-offs in edge model performance, communication efficiency, and cost of sequential communication. (Left)** Accuracy vs. edge model size, with the dashed line showing the GPT-4o model baseline. **(Middle)** Communication efficiency of Minion$\mathcal{S}$ with different local LMs, where larger models (7–8B) are more token-efficient. **(Right)** The trade-off between cost and quality across multiple rounds, where the x-axis represents the number of tokens processed by the *remote model*, and the y-axis shows the accuracy achieved. Point labels indicate the number of rounds of communication, with the dashed line at the top representing the GPT-4o model benchmark.

Here, we analyze how the design of Minion$\mathcal{S}$ affects cost and quality. Our main takeaways are:

- On average across three datasets, Minion$\mathcal{S}$ can recover 97.9% of the performance of remote-only systems while spending $5.7\times$ less;
- We identify protocol hyper-parameters that let us flexibly trade-off cost and quality;
- As local models grow stronger, Minion$\mathcal{S}$ becomes increasingly cost-effective.

We structure our analysis around three core design choices:

1. **Model choice** *How does the choice of local and remote model effect cost and quality?* We examine different model types and sizes for LocalLM and RemoteLM in Section 6.2.
2. **Scaling parallel workloads on-device** *How should we structure parallel workloads on the local device to maximize performance and minimize cost?* We highlight how scaling the local workloads can improve performance (Section 6.3) and study the effects on cost.
3. **Sequential communication protocol** *Can multiple rounds of communication improve quality? At what cost?* We explore this trade-off in Section 6.4.

Our findings are detailed in Sections 6.2, 6.3, and 6.4.

6.1 EXPERIMENTAL SETUP

Datasets and models We evaluate MINIONS on three benchmarks that are well suited for data-intensive reasoning: FINANCEBENCH, LONGHEALTH, and QASPER. FINANCEBENCH tests financial document understanding with complex reasoning over reports. LONGHEALTH focuses on tracking and interpreting longitudinal health records. QASPER assesses question answering over dense scientific papers. See Appendix B.0.1 for details. We use two open-source model families (LLAMA, QWEN2.5) as LocalLM and GPT-4O as RemoteLM (details in Appendix B.0.2).

6.2 MODEL CHOICE

This section explores the model requirements and generalization capabilities of MINIONS, examining the local model sizes necessary for effective collaboration, the sensitivity of the communication protocol across different local-remote model pairings, and the longitudinal evolution of MINIONS ' performance with advances in model capabilities over time.

What size do* LocalLM *have to be in order to be effective in* MINIONS*? Our results demonstrate that MINIONS starts being competitive with RemoteLM-only baseline at the 3B parameter model scale. When considering both the QWEN2.5 and LLAMA model families running locally, at 1B scale, MINIONS recovers 49.5% of the GPT-4O-only baseline performance, 3B scale recovers 93.4% and 8B recovers 97.9% accuracy (see Table 1 for more details).

How does the capacity of* LocalLM *affect the cost-accuracy tradeoff? In our system, LocalLM implicitly acts as an information encoder, optimizing the Information Bottleneck objective (Tishby et al., 2000) by compressing input context while preserving predictive information (see appendix D.2). To measure this, we analyze the tradeoff between remote "prefill" tokens (fewer tokens indicate greater compression) and accuracy (higher accuracy means better retention). Figure 4 shows that as LocalLM size increases, representations become more compressed and accurate, improving Information Bottleneck values. Larger LocalLM models trade local FLOPs for communication, with 7–8B models being 1.53× more token-efficient than 1B models. Additionally, the QWEN2.5 family follows a different tradeoff than LLAMA, yielding more compressed representations. This suggests that as small LMs improve, local-remote systems will become increasingly cost-efficient.

Is* MINIONS *sensitive to different local/remote pairs? We ask whether the communication protocol in MINIONS is invariant to changing the model types (*i.e.* LLAMA vs QWEN2.5 locally and LLAMA vs GPT-4O remotely). Our results indicate that MINIONS performs similarly with different local-remote LM combinations (see the Table 1): varying the LocalLM from QWEN2.5 to LLAMA-3.2, results in performances within $\pm$ .05 performance points (see Table 1). Furthermore, we find that holding the LocalLM fixed as LLAMA-3.2-3B and varying RemoteLM from GPT-4O to LLAMA-3.3-70B leads to similar overall performances within $\pm$ 0.07 points (see App. Table 2).

How have local / remote model capabilities changed over time, and what effects do they have on* MINIONS*? In Table 3, we provide a retrospective analysis demonstrating how the quality of MINIONS would have changed with model releases over time. From 2023 to 2025, the average performance of MINIONS with the best models available has improved from 0.26 to 0.66 (see App. Table 3). Interestingly, it was only in July 2024 — with the emergence of GPT4-TURBO and LLAMA-3.1-8B — that MINIONS could have come within 12% of the best frontier model performance at the time (see App. Table 3).

6.3 SCALING PARALLEL WORKLOADS ON-DEVICE

In MINIONS, there are three levers for maximizing local compute resources through parallelized, batched processing: (1) number of tasks per round, (2) number of samples taken per task, and (3) number of chunks. We ablate each, showing their impact on performance. We find that (1) and (3) are more cost effective ways of increasing performance.

How does the number of tasks per round affect performance? Increasing tasks per round proxies task decomposition, with more sub-tasks enhancing decomposition. Raising tasks from 1 to 16 boosts performance by up to 14 points but doubles $\mathrm{RemoteLM}$ prefill costs. Optimal task count varies by query and model, but exceeding 16 reduces performance.

How does scaling local samples affect performance? We explore whether increased sampling at an individual $\{$task, context$\}$ level improves performance. Increased sampling enables us to better utilize the available compute resources while improving task-level accuracy (Brown et al., 2024). Our results indicate that increasing the number samples from 1 to 32 can improve performance on average 7.4 points, but comes at the cost of $5\times$ the $\mathrm{RemoteLM}$ prefill costs. This being said, increasing sampling beyond 16 starts hurting task performance as the noise across samples is too large for the remote model to effectively distill the correct answer (Kuratov et al., 2024).

What effect does chunk size have on downstream performance? We test whether increasing local utilization by using more chunks per task improves performance. Our results indicate that increasing # of chunks per task (by decreasing the number of "pages" per chunk from 100 to 5) leads to an 11.7 point accuracy lift. However, this lift comes with a $2.41\times$ increase in prefill costs.

6.4 SCALING SEQUENTIAL COMMUNICATION

Both the MINION and MINION$\mathcal{S}$ communication protocols feature *sequential* communication: they allow for multiple rounds of exchange between the local and remote models.

Does performance improve as we increase the maximum number of rounds? At what cost? We vary the maximum communication rounds and find it is correlated with accuracy and cost (see fig. 4). By simply increasing the maximum number of rounds in MINION from 1 to 5, we enable a 8.5-point lift in average accuracy across the three tasks (with LLAMA-3.2 on-device). However, longer conversations also cost more: on FINANCEBENCH, each additional round of communication increases cost by 0.006 per query and accuracy by 4.2 accuracy points.

How should we maintain context between* MINION$\mathcal{S}$ *rounds? We experiment with two sequential protocol strategies: (1) simple retries and (2) scratchpad. See section 5 for details of these strategies. As shown in Figure 5, both strategies show consistent increases in both accuracy and cost when increasing the maximum number of rounds, with the scratchpad strategy achieving a slightly better cost-accuracy tradeoff. Notably, each additional round of communication with the scratchpad strategy leads to a larger improvement in accuracy (6.1 accuracy points) which are mostly offset by larger increases in cost (8.6 dollars).

6.5 RETRIEVAL AUGMENTED GENERATION IN THE CONTEXT OF LOCAL-REMOTE COMPUTE

In this section, we examine the relationship between local-remote collaboration (*e.g.*, MINION$\mathcal{S}$) and retrieval-augmented generation (RAG). These complementary techniques can be combined for different benefits.

In appendix E.3.1, we compare MINION$\mathcal{S}$, MINION, and RAG on two data-intensive reasoning tasks: one focused on extraction (FINANCEBENCH) and another on summarization (BOOOKSCORE). On FINANCEBENCH, RAG achieves similar or better cost-accuracy tradeoffs than MINION$\mathcal{S}$ but is less cost-effective than MINION. On BOOOKSCORE (appendix E.3.2), which requires integrating dispersed information, MINION$\mathcal{S}$ produces summaries comparable to a GPT-4O-only baseline, while RAG omits key plotlines, characters, and details.

7 DISCUSSION

Our study explores two protocols, MINION and MINION$\mathcal{S}$, for collaboration between on-device and cloud LMs, demonstrating that cloud computing costs can be reduced by 5.7–$30.4\times$ by effectively delegating tasks to local models. With increasingly powerful GPUs in consumer devices, users will rely less on cloud APIs, reducing operational costs while enabling complex local tasks like code refactoring and document analysis. MINION$\mathcal{S}$ highlights the promise of co-designing local and remote models to enhance efficiency, with the potential of moving beyond natural language to compressed real-valued representations in the future.

REFERENCES

Lisa Adams, Felix Busch, Tianyu Han, Jean-Baptiste Excoffier, Matthieu Ortala, Alexander Löser, Hugo JWL Aerts, Jakob Nikolas Kather, Daniel Truhn, and Keno Bressem. Longhealth: A question answering benchmark with long clinical documents. *arXiv preprint arXiv:2401.14490*, 2024.

Anthropic. The Claude 3 Model Family: Opus, Sonnet, Haiku. 2024. URL `https://www-cdn.anthropic.com/de8ba9b01c9ab7cbabf5c33b80b7bbc618857627/Model_Card_Claude_3.pdf`.

Simran Arora, Avanika Narayan, Mayee F Chen, Laurel Orr, Neel Guha, Kush Bhatia, Ines Chami, Frederic Sala, and Christopher Ré. Ask me anything: A simple strategy for prompting language models. *arXiv preprint arXiv:2210.02441*, 2022.

Simran Arora, Brandon Yang, Sabri Eyuboglu, Avanika Narayan, Andrew Hojel, Immanuel Trummer, and Christopher Ré. Language models enable simple systems for generating structured views of heterogeneous data lakes. *arXiv:2304.09433*, 2023.

Bradley Brown, Jordan Juravsky, Ryan Ehrlich, Ronald Clark, Quoc V Le, Christopher Ré, and Azalia Mirhoseini. Large language monkeys: Scaling inference compute with repeated sampling. *arXiv preprint arXiv:2407.21787*, 2024.

Yapei Chang, Kyle Lo, Tanya Goyal, and Mohit Iyyer. Booookscore: A systematic exploration of book-length summarization in the era of llms. *arXiv preprint arXiv:2310.00785*, 2023.

Lingjiao Chen, Matei Zaharia, and James Zou. Frugalgpt: How to use large language models while reducing cost and improving performance. *arXiv preprint arXiv:2305.05176*, 2023.

Lingjiao Chen, Jared Quincy Davis, Boris Hanin, Peter Bailis, Ion Stoica, Matei Zaharia, and James Zou. Are more llm calls all you need? towards scaling laws of compound inference systems. *arXiv preprint arXiv:2403.02419*, 2024a.

Zhuoming Chen, Avner May, Ruslan Svirschevski, Yuhsun Huang, Max Ryabinin, Zhihao Jia, and Beidi Chen. Sequoia: Scalable, robust, and hardware-aware speculative decoding. *arXiv preprint arXiv:2402.12374*, 2024b.

Pradeep Dasigi, Kyle Lo, Iz Beltagy, Arman Cohan, Noah A Smith, and Matt Gardner. A dataset of information-seeking questions and answers anchored in research papers. In *Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pp. 4599–4610, 2021.

Luc Devroye. Nonuniform random variate generation. *Handbooks in operations research and management science*, 13:83–121, 2006.

Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman, Akhil Mathur, Alan Schelten, Amy Yang, Angela Fan, et al. The Llama 3 Herd of Models. *arXiv preprint arXiv:2407.21783*, 2024. URL `https://arxiv.org/abs/2407.21783`.

Tom Gunter, Zirui Wang, Chong Wang, Ruoming Pang, Andy Narayanan, Aonan Zhang, Bowen Zhang, Chen Chen, Chung-Cheng Chiu, David Qiu, et al. Apple intelligence foundation language models. *arXiv preprint arXiv:2407.21075*, 2024.

Taicheng Guo, Xiuying Chen, Yaqi Wang, Ruidi Chang, Shichao Pei, Nitesh V Chawla, Olaf Wiest, and Xiangliang Zhang. Large language model based multi-agents: A survey of progress and challenges. *arXiv preprint arXiv:2402.01680*, 2024.

Kelvin Guu, Kenton Lee, Zora Tung, Panupong Pasupat, and Ming-Wei Chang. Retrieval augmented language model pre-training. In *Proceedings of the 37th International Conference on Machine Learning*, 2020.

Michael Hassid, Tal Remez, Jonas Gehring, Roy Schwartz, and Yossi Adi. The larger the better? improved llm code-generation via budget reallocation. *arXiv preprint arXiv:2404.00725*, 2024.

Pranab Islam, Anand Kannappan, Douwe Kiela, Rebecca Qian, Nino Scherrer, and Bertie Vidgen. Financebench: A new benchmark for financial question answering. *arXiv preprint arXiv:2311.11944*, 2023.

Gautier Izacard and Edouard Grave. Unsupervised dense information retrieval with contrastive learning. In *Advances in Neural Information Processing Systems*, 2021.

Dulhan Jayalath, James Bradley Wendt, Nicholas Monath, Sandeep Tata, and Beliz Gunel. Long-range tasks using short-context llms: Incremental reasoning with structured memories. *arXiv preprint arXiv:2412.18914*, 2024.

Hongpeng Jin and Yanzhao Wu. Ce-collm: Efficient and adaptive large language models through cloud-edge collaboration. *arXiv preprint arXiv:2411.02829*, 2024.

Vladimir Karpukhin, Barlas Oğuz, Sewon Min, Patrick Lewis, Ledell Wu, Sergey Edunov, Danqi Chen, and Wen-tau Yih. Dense passage retrieval for open-domain question answering. *arXiv preprint arXiv:2004.04906*, 2020.

Omar Khattab, Arnav Singhvi, Paridhi Maheshwari, Zhiyuan Zhang, Keshav Santhanam, Sri Vardhamanan, Saiful Haq, Ashutosh Sharma, Thomas T Joshi, Hanna Moazam, et al. Dspy: Compiling declarative language model calls into self-improving pipelines. *arXiv preprint arXiv:2310.03714*, 2023.

Yuri Kuratov, Aydar Bulatov, Petr Anokhin, Ivan Rodkin, Dmitry Sorokin, Artyom Sorokin, and Mikhail Burtsev. Babilong: Testing the limits of llms with long context reasoning-in-a-haystack. *arXiv preprint arXiv:2406.10149*, 2024.

Kenton Lee, Ming-Wei Chang, and Kristina Toutanova. Latent retrieval for weakly supervised open domain question answering. *arXiv preprint arXiv:1906.00300*, 2019.

Yaniv Leviathan, Matan Kalman, and Yossi Matias. Fast inference from transformers via speculative decoding. In *International Conference on Machine Learning*, pp. 19274–19286. PMLR, 2023.

Patrick Lewis, Ethan Perez, Aleksandra Piktus, Fabio Petroni, Vladimir Karpukhin, Naman Goyal, Heinrich Küttler, Mike Lewis, Wen-tau Yih, Tim Rocktäschel, et al. Retrieval-augmented generation for knowledge-intensive nlp tasks. *Advances in Neural Information Processing Systems*, 33: 9459–9474, 2020.

Chengshu Li, Jacky Liang, Andy Zeng, Xinyun Chen, Karol Hausman, Dorsa Sadigh, Sergey Levine, Li Fei-Fei, Fei Xia, and Brian Ichter. Chain of code: Reasoning with a language model-augmented code emulator. *arXiv preprint arXiv:2312.04474*, 2023.

Sachin Mehta, Mohammad Hossein Sekhavat, Qingqing Cao, Maxwell Horton, Yanzi Jin, Chenfan Sun, Seyed Iman Mirzadeh, Mahyar Najibi, Dmitry Belenko, Peter Zatloukal, et al. Openelm: An efficient language model family with open training and inference framework. In *Workshop on Efficient Systems for Foundation Models II@ ICML2024*, 2024.

Arvind Neelakantan, Tao Xu, Raul Puri, Alec Radford, Jesse Michael Han, Jerry Tworek, Qiming Yuan, Nikolas Tezak, Jong Wook Kim, Chris Hallacy, et al. Text and code embeddings by contrastive pre-training. *arXiv preprint arXiv:2201.10005*, 2022.

Charles Packer, Sarah Wooders, Kevin Lin, Vivian Fang, Shishir G Patil, Ion Stoica, and Joseph E Gonzalez. Memgpt: Towards llms as operating systems. *arXiv preprint arXiv:2310.08560*, 2023.

Pruthvi Patel, Swaroop Mishra, Mihir Parmar, and Chitta Baral. Is a question decomposition unit all we need? *arXiv preprint arXiv:2205.12538*, 2022.

Fabio Petroni, Tim Rocktäschel, Patrick Lewis, Anton Bakhtin, Yuxiang Wu, Alexander H. Miller, and Sebastian Riedel. Language models as knowledge bases? In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing*, 2019.

Stephen Robertson and Hugo Zaragoza. The probabilistic relevance framework: Bm25 and beyond. *Foundations and Trends in Information Retrieval*, 3:333–389, 01 2009. doi: 10.1561/1500000019.

Melisa Russak, Umar Jamil, Christopher Bryant, Kiran Kamble, Axel Magnuson, Mateusz Russak, and Waseem AlShikh. Writing in the margins: Better inference pattern for long context retrieval. *arXiv preprint arXiv:2408.14906*, 2024.

Jon Saad-Falcon, Adrian Gamarra Lafuente, Shlok Natarajan, Nahum Maru, Hristo Todorov, Etash Guha, E Kelly Buchanan, Mayee Chen, Neel Guha, Christopher Ré, et al. Archon: An architecture search framework for inference-time techniques. *arXiv preprint arXiv:2409.15254*, 2024.

Shreya Shankar, Aditya G Parameswaran, and Eugene Wu. Docetl: Agentic query rewriting and evaluation for complex document processing. *arXiv preprint arXiv:2410.12189*, 2024.

Kurt Shuster, Douwe Kiela, Ethan Perez, Harm de Vries, Jack Urbanek, Arthur Szlam, and Jason Weston. Retrieval augmentation reduces hallucination in conversation. In *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*, 2021.

Charlie Snell, Jaehoon Lee, Kelvin Xu, and Aviral Kumar. Scaling llm test-time compute optimally can be more effective than scaling model parameters. *arXiv preprint arXiv:2408.03314*, 2024.

Yifan Song, Guoyin Wang, Sujian Li, and Bill Yuchen Lin. The good, the bad, and the greedy: Evaluation of llms should not ignore non-determinism. *arXiv preprint arXiv:2407.10457*, 2024.

Naftali Tishby, Fernando C Pereira, and William Bialek. The information bottleneck method. *arXiv preprint physics/0004057*, 2000.

Junlin Wang, Jue Wang, Ben Athiwaratkun, Ce Zhang, and James Zou. Mixture-of-agents enhances large language model capabilities. *arXiv preprint arXiv:2406.04692*, 2024.

Tongshuang Wu, Michael Terry, and Carrie Jun Cai. Ai chains: Transparent and controllable human-ai interaction by chaining large language model prompts. In *Proceedings of the 2022 CHI conference on human factors in computing systems*, pp. 1–22, 2022.

Yangzhen Wu, Zhiqing Sun, Shanda Li, Sean Welleck, and Yiming Yang. An empirical analysis of compute-optimal inference for problem-solving with language models. *arXiv preprint arXiv:2408.00724*, 2024.

Daliang Xu, Hao Zhang, Liming Yang, Ruiqi Liu, Gang Huang, Mengwei Xu, and Xuanzhe Liu. Empowering 1000 tokens/second on-device llm prefilling with mllm-npu. *arXiv preprint arXiv:2407.05858*, 2024.

Zheming Yang, Yuanhao Yang, Chang Zhao, Qi Guo, Wenkai He, and Wen Ji. Perllm: Personalized inference scheduling with edge-cloud collaboration for diverse llm services. *arXiv preprint arXiv:2405.14636*, 2024.

Rongjie Yi, Xiang Li, Weikai Xie, Zhenyan Lu, Chenghua Wang, Ao Zhou, Shangguang Wang, Xiwen Zhang, and Mengwei Xu. Phonelm: an efficient and capable small language model family through principled pre-training. *arXiv preprint arXiv:2411.05046*, 2024.

Mert Yuksekgonul, Federico Bianchi, Joseph Boen, Sheng Liu, Zhi Huang, Carlos Guestrin, and James Zou. Textgrad: Automatic" differentiation" via text. *arXiv preprint arXiv:2406.07496*, 2024.

Kaiyan Zhang, Jianyu Wang, Ning Ding, Biqing Qi, Ermo Hua, Xingtai Lv, and Bowen Zhou. Fast and slow generating: An empirical study on large and small language models collaborative decoding. *arXiv preprint arXiv:2406.12295*, 2024a.

Yusen Zhang, Ruoxi Sun, Yanfei Chen, Tomas Pfister, Rui Zhang, and Sercan Ö Arik. Chain of agents: Large language models collaborating on long-context tasks. *arXiv preprint arXiv:2406.02818*, 2024b.

Zihan Zhou, Chong Li, Xinyi Chen, Shuo Wang, Yu Chao, Zhili Li, Haoyu Wang, Rongqiao An, Qi Shi, Zhixing Tan, et al. Llm x mapreduce: Simplified long-sequence processing using large language models. *arXiv preprint arXiv:2410.09342*, 2024.

A Extended Related Work

Orchestration of LMs Recent works attempt to improve long document processing by taking a divide-and-conquer approach akin to MINIONS. Instead of using single LM calls with the entire context, the task is decomposed into smaller tasks to be executed on chunks of context. (Zhang et al., 2024b; Zhou et al., 2024) use a predefined protocol for chunk processing (defined by a prompt). (Shankar et al., 2024) performs a more involved automated pipeline optimization (via agent-based rewrite directives). Crucially, none of the works study the cost-efficient interaction between a small local LM and large remote LM and instead focus exclusively on larger LMs (70B parameters and above). Moreover, they do not explore multi-round communication patterns for document analysis.

Long-context management techniques These works aim to improve *single LM* accuracy in long context tasks. (Russak et al., 2024) prefill the context using *chunks* of the document, summarize each chunk (using a predefined prompt), and aggregate the results. This improves accuracy and requires marginal additional computation. PRISM similarly (Jayalath et al., 2024) processes the context as a stream of chunks, and writes important information into a typed data structure which can be amended as needed. MemGPT (Packer et al., 2023) proposes a virtual memory paging system inspired by operating systems, where the LLM manages information across main context (akin to RAM) and external storage (akin to disk). When approaching context limits, the system actively decides what information to preserve and can later retrieve this information through paginated function calls. Orthogonally, other methods explore the usage of code for context management (Arora et al., 2023).

Cost-efficient multi-LLM Systems A plethora of recent works show that multiple LMs can collaborate on a task to improve both accuracy and efficiency (Guo et al., 2024). The most similar work is perhaps (Wang et al., 2024) which neither investigates LMs with with asymmetric capabilities nor optimizes for local compute efficiency.

Model routing techniques Our work studies a collaboration of LMs, and thus should be differentiated from model routing techniques (Chen et al., 2024a; 2023) that route a prompt to the appropriate single LM that can completely answer it using the full context. This is often done for cost reduction, identifying that simple tasks can be executed by smaller and cheaper LMs.

Compound LM systems Recent works explore the use of LMs as part of more elaborate pipelines that, retrieval models, tool use, and more. (Saad-Falcon et al., 2024; Khattab et al., 2023; Yuksekgonul et al., 2024) seeks to optimize the pipeline architecture and prompts using different approaches, which we do not pursue on this work.

Retrieval-Augmented Generation (RAG) RAG is a hybrid approach that integrates information retrieval into the text generation process, leveraging external knowledge sources to enhance the output of language models (LMs). Instead of relying solely on parametric memory, RAG reduces the number of tokens processed by an LM by first retrieving a subset of relevant documents or document chunks and appending them as context to the LM (Lewis et al., 2020; Karpukhin et al., 2020; Lee et al., 2019; Izacard & Grave, 2021; Guu et al., 2020). This retrieval step mitigates issues such as hallucination and knowledge staleness, which are common in traditional autoregressive models (Shuster et al., 2021; Petroni et al., 2019). We differ in two ways: first, our local LM can perform tasks beyond information extraction, such as summarization or reasoning. Second, by performing arbitrary tasks on document chunks, the small LM communicates its compact answer instead of the raw document chunk, which amounts to sending fewer tokens to remote.

Speculative decoding Speculative decoding (Leviathan et al., 2023; Zhang et al., 2024a; Chen et al., 2024b) techniques are addressing the different question of how to effectively sample from the distribution of a large LM by only sampling from smaller LM and using the large LM for cheaper, likelihood evaluation (using the "acceptance-complement algorithm" (Devroye, 2006)). It neither considers a collaboration between two LMs, nor attempts to minimize the communication between them.

On-device language models for privacy siyan2024papillon, zhang2024cogenesis attempt to prevent leaks of private information to a cloud-hosted LM API by mediating the communication with a local privacy-aware LM that removes private information from the prompt. While the local-remote LM setup bears resemblance to ours, we do not study the aspects of privacy, but rather focus on reducing cloud costs by delegating work to devices while maintaining accuracy. Moreover, we have additional focus on local runtime efficiency.

Local-remote systems Recent work has explored efficient routing patterns between local and remote computation for LM workloads, albeit without two models communicating or collaborating on a solution. (Jin & Wu, 2024) partition a single LLM with early layers on the edge and later layers in the cloud, routing to the cloud when confidence is low. (Yang et al., 2024) propose a complementary task scheduling framework that routes to cloud or local based on resource constraints and service requirements.

B Extended Description of Experimental Setup

B.0.1 Dataset Details

In this section we provide additional details on dataset preparation. In order to extend the context length of the problems in LongHealth and QASPER, we make a few modification to the dataset.

FinanceBench We filter the original FinanceBench to include only the numerical reasoning, resulting in a dataset of length 64. Each sample has an average context length of $142.9K(\pm 79224.32)$.

LongHealth In the original instantiation of the LongHealth dataset, each question is paired with a set of medical documents corresponding to a single patient. To increase the complexity of the dataset, we include medical documents from 10 other patients in the context. We evaluate over the entire dataset (400 problems) for results reported in Table 1. Each sample has an average context length of $120.1K(\pm 1, 237)$ tokens. For all ablations in Section 6, we use a fixed subset of 128 problems.

QASPER Similarly, in the QASPER dataset, the original dataset provides questions that are associated with a single scientific paper. In order to increase complexity, we include 10 other papers in the context. We evaluate over a random subset of 500 problems for results reported in Table 1. Each sample has an average context length of 54281 tokens (± 2403). For all ablations in Section 6, we use a fixed subset of 128 problems.

B.0.2 Model Details

Local Models. For Qwen2.5 we use the following models: Qwen2.5-1.5-Instruct, Qwen2.5-3B-Instruct, Qwen2.5-7B-Instruct. For Llama, we use the following models: Llama-3.2-1B-Instruct, Llama-3.2-3B-Instruct, Llama-3.1-8B-Instruct.

Remote Models. We use GPT-4o and Llama-3.2-70B-Instruct, Llama-3.1-70B-Instruct

All "local-only" and "remote-only" experiments are run with temperature of 0.2. For all Minion$\mathcal{S}$ experiments run in Table 1, we run the $\mathrm{RemoteLM}$ with a temperature of 0.0 and $\mathrm{LocalLM}$ with a temperature of 0.2 for FinanceBench and 0.00001 for QASPER and LongHealth.

C Extended Discussion of Cost Model

Here, we explain in detail the costs of the different communication protocols discussed in this paper—remote-only, Minion, and Minion$\mathcal{S}$—with a strong focus on the latency of these methods. This section is organized as follows:

- Section C.1: We review background on language model inference, to motivate our cost and latency models.
- Section C.2: We present mathematical models for the latency of the remote-only, Minion, and Minion$\mathcal{S}$ protocols.
- Section C.3: We present Proposition C.3, an upper bound on the total latency of Minion$\mathcal{S}$, relative to that of the remote-only model, demonstrating that Minion$\mathcal{S}$ is not much slower

than the naive approach of performing the full query in the cloud. As an example, we show that a Llama-8B model on a GTX-4090 GPU collaborating via MINION$\mathcal{S}$ with a Llama-405B model on a $8\times$H100 server is at most $4.75\times$ slower than the remote-only protocol.

C.1 BACKGROUND ON LANGUAGE MODEL INFERENCE

Language model inference consists of a sequence of forward passes through a model, one for prefill (*i.e.* input) followed by one for each additional token generated (*i.e.* output). At low/medium batch sizes, each forward pass after prefill is I/O bound, meaning the time it takes to load weights from memory exceeds the time it takes to actually compute the output. As the batch size increases, the computational cost of the forward pass eventually exceeds the I/O cost. Strikingly, for most models and hardware, this happens at a batch size > 100 (Leviathan et al., 2023; Chen et al., 2024b). As a result of this transition from being I/O bound to being compute bound, we can model (as is common in the literature) the cost of running a forward pass as a piecewise linear function $C_{\mathcal{M},\mathcal{E}}(n) = \max(\lambda, \alpha \cdot n + \beta)$ of the number of tokens n being processed. This is because for small n, the IO cost dominates (and is roughly constant as n grows), whereas at larger n the compute cost dominates and scales roughly linearly with n (assuming n is not *too* large).

In the cloud, the provider can batch generation requests from multiple users to keep hardware utilization high. Therefore, the cost of each output token is typically within a small multiple of the cost of each input token, and the total cost of processing the request scales as $n_{prefill} + \alpha \cdot n_{decode}$, for some small $\alpha \leq 5$.

On-device, we cannot assume we'll have enough concurrent user requests to form a large enough batch to achieve high utilization. As a result, the latency of a request does not scale linearly with the number of tokens. A single request can occur similar latency to hundreds run in parallel. As a result, tokens are a poor proxy for cost on-device and we instead measure latency in micro experiments (see Section 6.3).

C.2 LATENCY MODELS FOR ALL PROTOCOLS: REMOTE-ONLY, MINION, MINION$\mathcal{S}$

We now model the latency of each of these protocols (remote-only, MINION, MINION$\mathcal{S}$). We will then use these results in the following section to upper bound the latency of MINION$\mathcal{S}$ by a scalar multiple of the latency of the remote-only protocol.

First, we introduce the following assumptions and notation:

- We assume we have a local GPU (*e.g.* RTX-4090) with peak compute F_l (flops/sec), and peak bandwidth M_l (bytes/sec), and a remote GPU (*e.g.* H100) with peak compute F_r (flops/sec), and peak bandwidth M_r (bytes/sec),
- We also assume for now simple transformer architectures for both the local and remote models:
  - LocalLM: L_l layers, each with $8d_l^2$ params in MLP (Up/down projections each of size $d_l \times 4d_l$, and $4d_l^2$ parameters in the $W_{Q,K,V,O}$ projections. The total memory required for the (non-embedding/LM head) parameters is thus $P_l = 2 \cdot 12 L_l d_l^2$. For simplicity, we assume the memory for the LM head is small relative to P_l.
  - RemoteLM: Equivalent architecture to the LocalLM, but with L_r layers, d_r hidden dimension, and P_r total non-embedding/LM-head parameter memory (again assumed to be much greater than the number of LM head parameters).
- We model the number of input/output tokens of each protocol as follows, letting n denote the number of tokens in the original document:
  - **Remote-only**: n prefill tokens and n_{out}^r decode tokens. Note that we assume—here and below—that the number of tokens in the query is negligible relative to n. We assume $n \gg n_{out}^r$ so we can effectively ignore the KV-cache load time for the output tokens.
  - **MINION**: For LocalLM, we assume n prefill tokens and n_{out}^l decode tokens. For RemoteLM, we assume n_{out}^l prefill tokens, and n_{out}^r decode tokens. In the case of

multiple rounds of communication, the KV cache for the document can be stored to avoid recomputation.

  - **MINION$\mathcal{S}$**: For LocalLM, we assume n/c prefill tokens per chunk (c chunks total), and n^l_{out} decode tokens per job (though we assume only p fraction of output jobs do not abstain). For RemoteLM, we assume $J \cdot n^l_{out} \cdot p$ prefill tokens, and n^r_{out} decode tokens, letting $J = cks$ denote the total number of jobs in MINION$\mathcal{S}$ (c chunks, k instructions, s samples). In the case of multiple rounds of communication, the KV cache for each document chunk can be stored to avoid recomputation.

- Throughout, we use the fact that a $[m \times n] \cdot [n \times k]$ matmul takes $2 \cdot mnk$ flops, and assume model parameters are stored in half-precision (2 bytes/param).

We are now ready to present the latency models for the three protocols (remote-only, MINION, MINION$\mathcal{S}$).

C.2.1 REMOTE-ONLY

- **Prefill**: We are compute bound, so time is approximately given by $total\_flops/F_r$. We can break down $total\_flops$ into the matmuls (MLP up/down projections, and QKVO operations) and attention operations.
  - **Matmuls**: $2 \cdot 12nd^2$ per layer. Equivalent to a $[n \times d_r] \cdot [d_r \times 12d_r]$ matmul.
  - **Attention**: $2 \cdot n^2 d_r$ per layer. Equivalent to $[n \times d_r] \cdot [d_r \times n]$ matmul.
  - **Time**: $L_r \cdot (24nd_r^2 + 2n^2 d_r)/F_r = (nP_r + 2L_r d_r n^2)/F_r$.
- **Decode**: We are memory bound (batch size 1 for Minion), so time is approximately given by $total\_memory/M_r$ per decode step. We can break down $total\_memory$ into model parameters and KV cache.
  - **Model parameters**: $2 \cdot 12d_r^2$ bytes per layer.
  - **KV-cache**: $2 \cdot 2nd_r$ bytes per layer (K and V are each $[n \times d]$ matrices).
  - **Time**: $L_r \cdot n^r_{out} \cdot (24d_r^2 + 4nd_r)/M_r = n^r_{out}(P_r + 4L_r d_r n)/M_r$.

Total time is given by the sum of prefill and decode times:

$$T_{remote} = \frac{nP_r + 2L_r d_r n^2}{F_r} + \frac{n^r_{out}(P_r + 4L_r d_r n)}{M_r}$$

C.2.2 MINION

The latency of the LocalLM in the MINION protocol can be modeled equivalently to the latency of the remote-only protocol, but replacing the remote parameters with the corresponding local ones. Thus, total local latency is:

$$T^{\text{MINION}}_{local} = \frac{nP_l + 2L_l d_l n^2}{F_l} + \frac{n^l_{out}(P_l + 4L_l d_l n)}{M_l}$$

The total remote latency can also be expressed using these same equations, but with n^l_{out} prefill tokens, and n^r_{out} decode tokens.

$$T^{\text{MINION}}_{remote} = \frac{n^l_{out}P_r + 2L_r d_r (n^l_{out})^2}{F_r} + \frac{n^r_{out}(P_r + 4L_r d_r n^l_{out})}{M_r}$$

C.2.3 MINION$\mathcal{S}$

The LocalLM latency of the MINION$\mathcal{S}$ protocol has some important differences from the MINION protocol—the prefill computation avoids cross-chunk attention (which saves time), while the decode operations can actually be compute bound if batching of the different jobs is done. We review these details below:

- **Prefill**: We are compute bound, so time is approximately given by $total\_flops/F$. We can break down $total\_flops$ into the matmuls (MLP up/down projections, and QKVO operations) and attention operations.

  - **Matmuls**: $2 \cdot 12nd_l^2$ per layer. Equivalent to $c\;[n_c \times d_l] \cdot [d_l \times 12d_l]$ matmuls (where $n_c = n/c$).
  - **Attention**: $2 \cdot cn_c^2 d_l = 2 \cdot c\;(n/c)^2 d_l = 2n^2 d_l/c$ per layer. Equivalent to $c\;[n_c \times d_l] \cdot [d_l \times n_c]$ matmuls.
  - **Time**: $L_l \cdot (24nd_l^2 + 2n^2 d_l/c)/F = (nP_l + 2L_l d_l n^2/c)/F$.

- **Decode**: We will now assume we are **compute bound during decode**, because we have many jobs (ks) per chunk, and many chunks (c) per document, which we can batch together. Thus, time is approximately given by $total\_flops/F_l$ per decode step. We can break down $total\_flops$ into matmuls and attention. The flops below are per job, per output token (so for total flops we will multiply by $n_{out}^l \cdot pcks$):
  - **Matmuls**: $2 \cdot 12d_l^2$ per layer. Equivalent to a $[1 \times d_l] \cdot [d_l \times 12d_l]$ matmul.
  - **Attention**: $2 \cdot n_c d_l = 2d_l\, n/c$ per layer. Equivalent to $[1 \times d_l] \cdot [d_l \times n_c]$ matmul.
  - **Time**: $L_l \cdot n_{out}^l \cdot pcks \cdot (24d_l^2 + 2d_l n/c)/F = n_{out}^l \cdot pcks \cdot (P_l + 2L_l d_l n/c)/F$.

The *total local latency* for MINION$\mathcal{S}$ is given by the sum of prefill and decode times:

$$T_{local} = \frac{nP_l + 2L_l d_l n^2/c}{F_l} + \frac{n_{out}^l \cdot pcks \cdot (P_l + 2L_l d_l n/c)}{F_l}.$$

The *total remote latency* for MINION$\mathcal{S}$ can be expressed using the same equations as MINION, but with $pcks \cdot n_{out}^l$ prefill tokens, and n_{out}^r decode tokens.

$$T_{remote} = \frac{(pcks \cdot n_{out}^l)P_r + 2L_r d_r (pcks \cdot n_{out}^l)^2}{F_r} + \frac{n_{out}^r (P_r + 4L_r d_r (pcks \cdot n_{out}^l))}{M_r}$$

C.3 MINION$\mathcal{S}$ VS. REMOTE-ONLY COMPARISON

Assume $n_{out}^l \cdot pcks = an$, for some $a < 1$, and that $F_{r,l}$, $d_{r,l}$, and $L_{r,l}$ are all as defined in Appendix C.2. In this case, we can show that the ratio of total latency of MINION$\mathcal{S}$ vs. the remote-only protocol is upper-bounded by the following expression:

$$\frac{T_{remote} + T_{local}}{T_{remote}} \quad < \quad 1 + \big(1 + a\big) \cdot \frac{F_r}{F_l} \cdot \frac{L_l d_l}{L_r d_r}.$$

Let's assume $n_{out}^l \cdot pcks = an$, for some $a < 1$.

$$\begin{aligned}
T_{local} \quad &= \quad \frac{nP_l + 2L_l d_l n^2/c}{F_l} + \frac{an \cdot (P_l + 2L_l d_l n/c)}{F_l} \\
&< \quad \big(1 + a\big) \cdot \frac{nP_l + 2L_l d_l n^2/c}{F_l} \\
T_{remote} \quad &= \quad \frac{(an)P_r + 2L_r d_r (an)^2}{F_r} + \frac{n_{out}^r (P_r + 4L_r d_r (an))}{M_r} \\
&< \quad a\bigg(\frac{nP_r + 2L_r d_r n^2}{F_r} + \frac{n_{out}^r 4L_r d_r n}{M_r}\bigg) + \frac{n_{out}^r P_r}{M_r} \\
T_{remote} \quad &= \quad \frac{nP_r + 2L_r d_r n^2}{F_r} + \frac{n_{out}^r (P_r + 4L_r d_r n)}{M_r}
\end{aligned}$$

Thus, it is easy to see that $\frac{T_{remote}}{T_{remote}} < 1$. Now let's look at $\frac{T_{local}}{T_{remote}}$, and show it is upper bounded by a constant:

$$\begin{aligned}
\frac{T_{local}}{T_{remote}} &< \frac{\left(1+a\right)\cdot\frac{nP_l+2L_ld_ln^2/c}{F_l}}{\frac{nP_r+2L_rd_rn^2}{F_r}} \\
&= \left(1+a\right)\cdot\frac{F_r}{F_l}\cdot\frac{nP_l+2L_ld_ln^2/c}{nP_r+2L_rd_rn^2} \\
&\leq \left(1+a\right)\cdot\frac{F_r}{F_l}\cdot\max\left(\frac{P_l}{P_r},\frac{L_ld_l}{L_rd_rc}\right) \\
&= \left(1+a\right)\cdot\frac{F_r}{F_l}\cdot\max\left(\frac{L_ld_l^2}{L_rd_r^2},\frac{L_ld_l}{L_rd_rc}\right) \\
&< \left(1+a\right)\cdot\frac{F_r}{F_l}\cdot\frac{L_ld_l}{L_rd_r}.
\end{aligned}$$

Thus, combining the above two results we can see that:

$$\frac{T_{remote}+T_{local}}{T_{remote}} < 1+\left(1+a\right)\cdot\frac{F_r}{F_l}\cdot\frac{L_ld_l}{L_rd_r}.$$

Real example: Let's assume that the local GPU is a RTX 4090 ($F_l \approx 160$ TFLOPS), the remote server is a full node of 8 H100s ($F_r \approx 8000$ TFLOPS across full node), the local model is Llama-8B ($L_l = 32$, $d_l = 4096$), and the remote model is Llama-405B ($L_l = 126$, $d_l = 16384$). Furthermore, let's assume $a \approx 0.2$, which is actually a bit larger than we see in practice. In this case:

$$\begin{aligned}
1+\left(1+a\right)\cdot\frac{F_r}{F_l}\cdot\frac{L_ld_l}{L_rd_r} &\approx 1+1.2\cdot\frac{8000}{160}\cdot\frac{32\cdot 4096}{126\cdot 16384} \\
&\approx 1+1.2\cdot 50\cdot\frac{1}{16} \\
&= 4.75.
\end{aligned}$$

Note that if we perform multiple rounds of MINION$\mathcal{S}$, this ratio gets multiplied by at most the number of rounds, though as mentioned previously, we can save time by only performing prefill on all the document chunks in the first round.

D EXTENDED DISCUSSION OF METHODS

D.1 EXTENDED DESCRIPTION OF MINION

In this section, we describe MINION, a baseline local-remote communication protocol. We ask whether we can reduce remote prefill tokens, and thus cost, by simply orchestrating a free-form conversation between the LocalLM and the RemoteLM in which only the LocalLM has direct access to the context $\mathbf{c}$.

The protocol proceeds with initialization step followed by a simple correspondence between the two models, which terminates when the remote model can answer the question or a maximum iteration limit is reached.

Iteration $i = 1$: Initialize. The RemoteLM receives the task query $\mathbf{q}$ along with a system prompt $\mathbf{p}_{\text{remote}}$ that instructs it to interact with a small LM that has full access to context. It outputs a first message $\mathbf{m}^{(1)}_{\text{remote}}$:

$$\mathbf{m}^{(1)}_{\text{remote}} \sim \text{RemoteLM}(\mathbf{q}, \mathbf{p}_{\text{remote}})$$

The message is then provided to LocalLM, along with the full context $\mathbf{c}$, the query $\mathbf{c}$, and a minimal system prompt $\mathbf{p}_{\text{local}}$ that instructs it to answer questions on the context:

$$\mathbf{m}^{(1)}_{\text{local}} \sim \text{LocalLM}(\mathbf{m}^{(1)}_{\text{remote}}, \mathbf{q}, \mathbf{p}_{\text{local}}, \mathbf{c})$$

Local Model	Remote Model	Release Date	Accuracy (Longhealth)	Accuracy (QASPER)	Accuracy (Finance)
llama-3B	gpt-4o	May 2024	0.7025	0.598	0.7826
llama-3B	gpt-4-turbo	April 2024	0.6247	0.614	0.6304
llama-3B	gpt-3.5-turbo-0125	Jan 2024	0.2157	0.4314	0.1707
llama-3B	gpt4o-mini	July 2024	0.6275	0.568	0.6522
llama-3B	llama3-70B-Instruct-Turbo	April 2024	0.3525	0.144	0.1818
llama-3B	llama3.1-70B-Instruct-Turbo	July 2024	0.6193	0.514	0.4348
llama-3B	llama3.3-70B-Instruct-Turbo	December 2024	0.6658	0.534	0.6739

Table 2: Accuracy Results for Longhealth, QASPER, and Finance across Various Models

Iteration $i > 1$. Step 1: Message from remote to local. $\mathrm{RemoteLM}$ consumes the conversation history and outputs new messages:

$$\mathbf{m}_{\text{remote}}^{(i)} \sim \mathrm{RemoteLM}(\mathbf{m}_{\text{remote}}^{(:i-1)}, \mathbf{m}_{\text{local}}^{(:i-1)}, \mathbf{q}, \mathbf{p}_{\text{remote}})$$

In its message, $\mathrm{RemoteLM}$ indicates whether it has sufficient information to terminate the loop and answer the question, or alternatively raises additional questions.

Step 2: Message from local to remote $\mathrm{LocalLM}$ consumes the latest remote message and conversation history, and outputs $\mathbf{m}_{\text{local}}^{(i)}$.

$$\mathbf{m}_{\text{local}}^{(i)} \sim \mathrm{LocalLM}(\mathbf{m}_{\text{remote}}^{(:i-1)}, \mathbf{m}_{\text{local}}^{(:i-1)}, \mathbf{q}, \mathbf{p}_{\text{local}}, \mathbf{c})$$

We then increment the iteration i and loop back to **Step 1** until the break condition is met or we reach a maximum number of iterations.

D.2 Information Bottleneck Perspective

How does local model capacity affect the cost-accuracy tradeoff?

The Information Bottleneck (IB) principle (Tishby et al., 2000) provides a useful analogy. One communication round of a local-remote system does as follows:

$$\mathbf{z} \sim p(\mathbf{z}|\mathbf{c}) \quad \text{[Extract info. from context]}$$
$$\mathbf{y} \sim p(\mathbf{y}|\mathbf{z}) \quad \text{[Predict outcome from extracted info]}$$

The IB principle seeks to find a $p(\mathbf{z} \mid \mathbf{c})$, our LocalLM, as follows:

$$\min_{p(\mathbf{z}|\mathbf{c})} \Big[I(C;Z) \; - \; \beta\, I(Z;Y) \Big], \tag{3}$$

i.e. find a mapping that forces the latent representation to be maximally informative of the label $I(Z;Y)$ and minimally informative of the input $I(C;Z)$, with a tradeoff parameter β. Here, we do not optimize the mapping $p(\mathbf{z} \mid \mathbf{c})$ but instead only get to choose it by setting $\mathrm{LocalLM}$.

Since we cannot compute these quantities in closed form for nonlinear distributions over tokens, we use (coarse) empirical proxies as follows. As a proxy for $I(C;Z)$, we compute the number of prefill tokens sent to $\mathrm{RemoteLM}$, capturing the intuition that more tokens carry more information on the input. $I(Z;Y)$ is estimated as the average accuracy of the local-remote system, quantifying the preservation of task-relevant information in $\mathbf{z}$. While these proxies do not exactly match the underlying mutual informations, they capture the core tension of compressing the input vs. preserving predictive power.

We plot these quantities in Figure **??**. We find that across both QWEN2.5 and LLAMA model families, as we increase $\mathrm{LocalLM}$ size, we send fewer tokens to $\mathrm{RemoteLM}$ ($\approx I(C;Z) \downarrow$), and improve accuracy ($\approx I(Z;Y) \downarrow$). We find that LLAMA has higher $\approx I(C;Z)$ and higher $\approx I(Z;Y)$.

E Extended Results

E.1 Model Analysis

We include additional experiment results from Section 6.2. In Table 2 we show the effects of varying $\mathrm{RemoteLM}$ on MINION$\mathcal{S}$. In Table 3, we show the performance of MINION$\mathcal{S}$ using the best in-class models at the time (from late 2023 to late 2024).

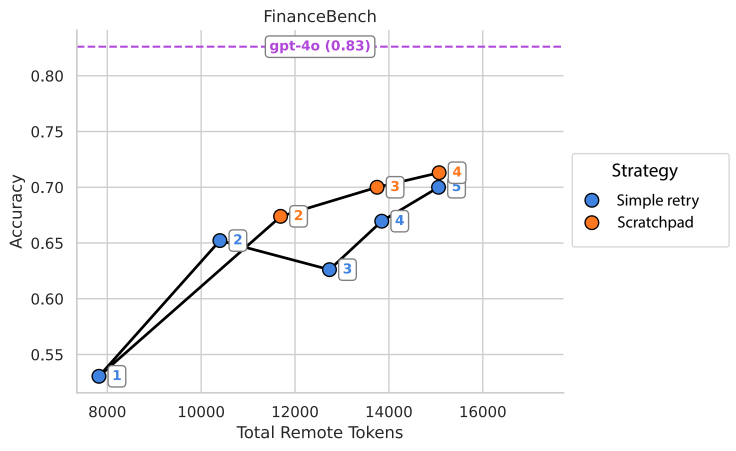


Figure 5: **Comparing strategies for maintaining context between MINIONS rounds.** The x-axis represents the number of tokens processed by the *remote model*, while the y-axis shows the accuracy achieved.

Local Model	Remote Model	Accuracy (Longhealth)	Accuracy (QASPER)	System Date
Llama-2-7b-chat-hf	gpt-4-1106-preview	0.340	0.178	November 2023
Llama-3.1-8B-Instruct	gpt-4-turbo	0.645	0.528	April 2024
Llama-3.1-8B-Instruct	gpt-4o	0.740	0.582	July 2024
—	gpt-4-turbo	0.768	0.391	April 2024

Table 3: Point in time results for MINIONS configurations with best-in-class LocalLM and RemoteLM

E.2 MINION LocalLM ANALYSIS

Total Chunks In-Context	Accuracy
1	0.59375
16	0.53906
32	0.50000
64	0.48438
128	0.46094

Table 4: Accuracy vs. Number of Chunks in Context
Each chunk has 512 tokens.

We perform an empirical analysis evaluating the robustness of LocalLM. We perform experiments to evaluate two axes of model capabilities: (1) ability to reason over long contexts and (2) ability to solve multi-part queries. To test (1) and (2) we curate a synthetic dataset built over the FINANCEBENCH dataset wherein we use GPT-4O to construct an extraction based question-answering dataset over chunks (length 512 tokens) of documents in the FINANCEBENCH dataset. We then construct two settings evaluating over LLAMA-3.2-3B-Instruct.

Long Context Reasoning: To evaluate long-context reasoning, we concatenate between {1,16,32,64,128} chunks to construct the context. At least one chunk in the concatenated context contains the ground truth result. As seen in Table 4, increasing the context length from 512 to 65.5K tokens leads to a 13 point drop in accuracy.

Multi-step Queries To evaluate the ability of LocalLM to fulfill multi-step queries, we construct queries that have between {1,2,3,4} sub-tasks. Our results indicate increasing from 1 to sub-tasks leads to a 56.3 point drop in accuracy (see Table 5).

Number of Sub Tasks	Accuracy
1	0.70313
2	0.39844
3	0.19531
4	0.14844

Table 5: Accuracy vs. Number of Sub Tasks

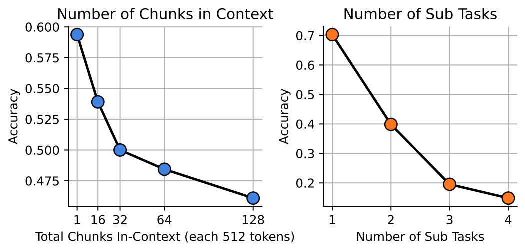


Figure 6: **Analysis of small LM limitations**. Evaluation of LLAMA-3.2-3B on simple extraction tasks (see Section E.2). **(Left)** Performance drops significantly as context length increases. **(Right)** Increasing sub-task complexity reduces performance, with fewer sub-tasks yielding better results.

E.3 RELATIONSHIP WITH RETRIEVAL-AUGMENTED GENERATION

In this section, we discuss the relationship between local-remote collaboration and retrieval-augmented generation (RAG), a technique that reduces the number of tokens processed by an LM by retrieving a subset of relevant documents or chunks LM Lewis et al. (2020); Karpukhin et al. (2020); Lee et al. (2019).

Retrieval-augmented generation and local-remote collaboration (*e.g.* MINION$\mathcal{S}$) are complementary techniques. They both provide a means to reduce cost by providing an LLM with a partial view of a large context. But, as we discuss below, they also have different error profiles and can be used in conjunction to improve performance.

E.3.1 COMPARISON OF MINION$\mathcal{S}$ AND RAG ON FINANCEBENCH

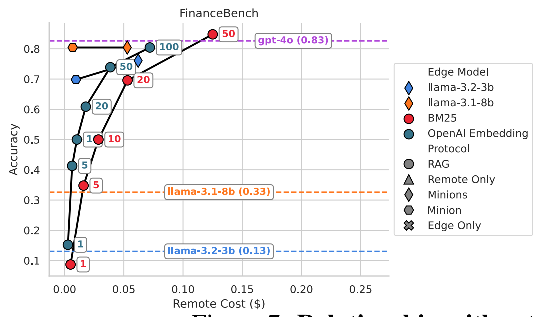
In Figure 7 (left), we plot the quality-cost trade-off on FINANCEBENCH for local-remote systems (MINION and MINION$\mathcal{S}$) and RAG systems using BM25 and OpenAI's `text-embedding-3-small` embeddings Robertson & Zaragoza (2009); Neelakantan et al. (2022). For RAG, we use a chunk size of 1000 characters, which we found to be optimal for this dataset after sweeping over chunk sizes with the BM25 retriever (see Figure 7 (center)). We show how a simple hyperparameter (number of retrieved chunks provided to the remote model) allows us to trade off quality of the RAG system for remote cost. Furthermore, we note that when the BM25 RAG system provides 50 or more chunks of the document to the remote model, it exceeds the performance of the remote model with the full context. This likely indicates that RAG helps in minimizing distractions from the long context. For FINANCEBENCH, when compared to MINION$\mathcal{S}$, the RAG system with OpenAI embeddings reaches similar points in the quality-cost trade-off space. Interestingly however, none of the RAG configurations are are able to match the quality of MINION at the same low cost.

E.3.2 COMPARISON OF MINION$\mathcal{S}$ AND RAG (EMBEDDINGS + BM25) ON SUMMARIZATION TASKS

RAG is a very suitable approach for FINANCEBENCH, since all of the questions heavily rely on information extraction from specific sections of financial statements. However, RAG will not be suitable for a summarization task, unlike small LMs. Therefore, we use the long-document summarization dataset, BOOOOKSCORE (Chang et al., 2023). BOOOOKSCORE which contains a set of 400 books published between 2023-2024. The average story length in BOOOOKSCORE is 128179

Protocol	Local Model	Remote Model	FINANCEBENCH				LONGHEALTH				QASPER		
			Acc.	Cost	In Tok. (1k)	Out Tok. (1k)	Acc.	Cost	In Tok. (1k)	Out Tok. (1k)	Acc.	Cost	In Tok. (1k
Remote Only	—	GPT-4O	0.826	\$0.261	103.04	0.32	0.748	\$0.301	120.10	0.07	0.598	\$0.137	54.40
Local Only	LLAMA-8B	—	0.326	\$0.000	0.00	0.00	0.468	\$0.000	122.58	0.07	0.538	\$0.000	54.41
Local Only	LLAMA-1B	—	0.000	\$0.000	0.00	0.00	0.115	\$0.000	122.58	0.07	0.000	\$0.000	54.41
Local Only	LLAMA-3B	—	0.130	\$0.000	0.00	0.00	0.345	\$0.000	122.58	0.08	0.164	\$0.000	54.41
Local Only	QWEN-3B	—	0.087	\$0.000	0.00	0.00	0.177	\$0.000	31.24	0.08	0.156	\$0.000	32.58
aive	LLAMA-8B	GPT-4O	0.804	\$0.007	0.88	0.46	0.635	\$0.010	1.85	0.50	0.450	\$0.007	0.92
aive	LLAMA-3B	GPT-4O	0.698	\$0.010	1.74	0.52	0.482	\$0.009	1.56	0.47	0.372	\$0.011	2.26
aive	QWEN-3B	GPT-4O	0.217	\$0.029	8.28	0.82	0.281	\$0.021	5.70	0.68	0.210	\$0.035	10.51
MINIONS	LLAMA-8B	GPT-4O	0.804	\$0.053	15.99	1.29	0.740	\$0.054	18.96	0.65	0.582	\$0.019	5.10
MINIONS	LLAMA-3B	GPT-4O	0.726	\$0.079	24.67	1.77	0.703	\$0.057	20.11	0.66	0.558	\$0.020	5.62
MINIONS	QWEN-3B	GPT-4O	0.783	\$0.059	17.20	1.56	0.645	\$0.043	14.43	0.65	–	–	–
MINIONS	QWEN-7B	GPT-4O	–	–	–	–	–	–	–	–	0.600	\$0.015	3.44

Table 6: **Accuracy and cost of local-remote systems.** Evaluation of cost and accuracy on the FINANCEBENCH Islam et al. (2023), LONGHEALTH Adams et al. (2024), and QASPER Dasigi et al. (2021). The table compares two edge-remote communication protocols — Naïve (section 4) and MINIONS (section 5) — alongside edge-only and remote-only baselines. Three different edge models are considered (LLAMA-8B, LLAMA-3B, QWEN-3B, and LLAMA-1B) and a remote model (GPT-4O). Accuracy (Acc.) is the fraction of correct predictions across the dataset. Cost (USD) is the average cost in USD per query in the dataset computed. Costs are incurred for any calls to the remote model at GPT-4O rates (January 2025: \$2.50 per million input tokens and \$10.00 per million output tokens). We assume that running the edge model is free; see section 3 for details on the cost model. In tokens is the number of input (*i.e.* prefill) tokens sent to the remote model. Out tokens is the number of output (*i.e.* decode) tokens generated from the remote model. Both values are shown in thousands.



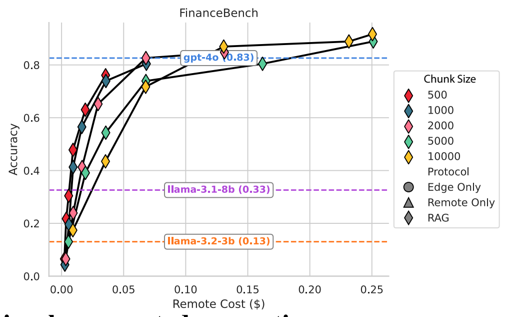


Figure 7: **Relationship with retrieval-augmented generation.**

tokens with a max of 401486 tokens and a minimum of 26926 tokens. We utilize both MINIONS, RAG (w/Embeddings + BM25), and GPT-4O only to complete the task. We describe the set-up for all three approaches next.

MINIONS for summarization In applying MINIONS to the task, the LocalLM (LLAMA-3.2-3B-Instruct) provides summaries on chunks of the original text, passing a list of chunk summaries to the RemoteLM (GPT-4O). RemoteLM produces the final summary.

RAG (Embedding) for summarization In our embedding-based RAG approach, we use the OpenAI TEXT-EMBEDDING-3-SMALL to embed chunks of the original text (of length 5000 characters) and we retrieve the top-15 most relevant chunks using the query "Summarize the provided text". We then prompt GPT-4O to generate a complete summary over the retrieved chunks.

RAG (BM25) for summarization In our BM25-based RAG approach, we use the BM25 to retrieve chunks of the original text (of length 5000 characters) based on the query: "Summarize the provided text". We retrieve the top-15 most relevant chunks and prompt GPT-4O to produce a final summary over the retrieved chunks. We choose top-15 to ensure the number of tokens passed up by the baseline is comparable with those passed up by MINIONS.

GPT-4o In our final baseline, we use GPT-4O alone to create the story summaries. For texts that extend beyond the 128K context length window, we truncate the stories.

Evaluation

- **Qualitative** In Table 8 we provide samples outputs from each of the 4 methods described above. We highlight major events in red, themes in green, locations in blue and names in indigo. The samples demonstrate that amongst all the methods, MINION$\mathcal{S}$ outputs contain the most entity mentions and story specific details. Moreover, when compared to GPT-4O-only RemoteLM, MINION$\mathcal{S}$ is $9.3\times$ more efficient — 11,500 versus the full 108,185 prefill tokens.

  The summaries from MINION$\mathcal{S}$ are generally $1.3\times$ longer and more verbose than the RAG systems' summaries, likely indicating that the former is more effective at "passing forward" salient information. Moreover, RAG systems' summaries are missing the main arc of the narrative in favor of what seems an assortment of facts.
- **Quantitative** We additionally perform a quantitative analysis of the generated summaries using a LLM-as-a-judge framework. As an evaluator, we use the CLAUDE-3.5-SONNET model, to avoid any biases between the evaluator and the supervisor model. We prompt the model with the generated summary, ground truth summary (gpt4-4096-inc-cleaned) provided from the original BOOOOKSCORE generations, and a grading rubric (see Figure 8). The rubric evaluates 7 criteria: coherence, relevance, conciseness, comprehensiveness, engagement & readability, accuracy, and thematic depth. We prompt CLAUDE-3.5-SONNET to generate a score (1-5) for each of the criteria and average the scores. We find that summaries generated by MINION$\mathcal{S}$ score comparably with GPT4O-only generated summaries, while RAG based baslines perform worse. Our results can be found in Table 7.

Evaluation Rubric for Summaries

1. **Coherence (1-5):** Summary is logically structured, with clear connections between events, avoiding abrupt jumps or inconsistencies.
2. **Relevance (1-5):** Accurately reflects key themes, events, and characters, focusing on essential details without unnecessary plot points.
3. **Conciseness (1-5):** Thorough yet avoids excessive detail, presenting necessary information without redundancy.
4. **Comprehensiveness (1-5):** Covers all major characters, events, and themes, ensuring a complete overview without omissions.
5. **Engagement & Readability (1-5):** Engaging and easy to read, with well-constructed sentences and clear, precise language.
6. **Accuracy (1-5):** Stays true to the book's storyline, themes, and tone, with correct details, names, and events.
7. **Thematic Depth (1-5):** Identifies underlying themes and messages, providing insights into conflicts, motivations, and resolutions.

Figure 8: Evaluation Rubric for Summaries

Method	Score
MINION$\mathcal{S}$	3.01
GPT4o	3.06
RAG (BM25)	2.48
RAG (Embedding)	2.38

Table 7: Comparison of Methods and Rubric Scores

Table 8: Comparative Analysis of Story Summaries Across Different Methods

Story Title	**Minions**	**GPT-4o**	**BM25**	**Embedding**
A Likely Story	The novel follows the complex dynamics of the Manning family, focusing on Isabelle Manning, a struggling writer grappling with the legacy of her famous father, Ward Manning, and the recent death of her mother, Claire. As Isabelle navigates her career and personal life, she uncovers family secrets, including her father's plagiarism and her mother's hidden manuscript, which challenge her identity and ambitions. The narrative explores themes of fame, family expectations, and personal growth, culminating in Isabelle's journey towards self-discovery and independence. Set against the backdrop of New York City and Sag Harbor, the story delves into the intricacies of familial relationships and the pursuit of creative fulfillment.	Isabelle Manning, daughter of a famous author, discovers an unfinished manuscript by her late mother, Claire. She completes and publishes it as her own, leading to success but also guilt and exposure. Her father, Ward, reveals he sabotaged her first book to teach her resilience, deepening her sense of betrayal. Ultimately, Isabelle confesses her actions, seeks redemption, and begins to redefine her identity beyond her father's shadow.	Isabelle Manning, overshadowed by her famous father Ward, receives a threatening text, "I KNOW WHAT YOU DID," hinting at a secret tied to her book, "Underpainting." Meanwhile, her mother Claire's past support and her father's hidden illness add layers to Isabelle's challenges.	In a narrative spanning New York and Sag Harbor, Claire Cunningham grapples with her identity beyond being a Vassar graduate, while her daughter Isabelle faces personal and professional challenges, including her father's illness and her own writing struggles.
All the Dangerous Things	Isabelle Drake, a woman grappling with the traumatic disappearance of her son Mason, navigates a complex web of grief, guilt, and suspicion. As she becomes entangled with true crime enthusiasts and investigators, including podcast host Waylon and Detective Dozier, Isabelle's quest for truth reveals unsettling family secrets and personal betrayals. Her journey is marked by strained relationships, particularly with her ex-husband Ben and his connections to other women, including Valerie and Allison. Throughout the narrative, themes of motherhood, mental health, and societal judgment are explored, culminating in a deeper understanding of the pressures and expectations faced by women.	Isabelle Drake, plagued by insomnia and guilt, is desperate to find her missing son, Mason. She suspects her husband, Ben, and his new partner, Valerie. With Waylon's help, she discovers Abigail Fisher, manipulated by Valerie, took Mason believing she was rescuing him.	The narrative follows Isabelle, dealing with Mason's disappearance. She works with podcaster Waylon, uncovering links to Ben's deceased wife, Allison.	Isabelle, struggling with grief and insomnia, joins a grief counseling group. She meets Valerie and collaborates with Waylon, but becomes wary after finding unsettling information on his laptop.

Continued on next page

Continued from previous page

Story Title	**Minions**	**GPT-4o**	**BM25**	**Embedding**
A Living Remedy: A Memoir	Nicole Chung, a Korean American adoptee, reflects on her complex relationships with her adoptive parents, her identity, and the challenges of navigating life as a minority in a predominantly white community in Oregon. Her memoir explores themes of family, loss, and resilience, as she recounts her father's death from kidney failure, and her mother's battle with cancer. Amidst these personal challenges, Chung grapples with her own grief, financial struggles, and the impact of the COVID-19 pandemic, while finding solace in her family, faith, and writing. Her journey is marked by a deep appreciation for her parents' sacrifices, the support of her husband and children, and the enduring legacy of love and forgiveness instilled by her mother.	Nicole Chung's memoir explores her journey after the loss of her adoptive parents. As a Korean adoptee, she reflects on family's financial struggles, parents' health battles, and their deaths' impact on her identity. She finds solace in writing and her own family.	The protagonist struggles with visiting her dying mother during the COVID-19 pandemic. The story explores grief, family responsibility, and cherishing life amidst adversity.	A woman reflects on her parents' illnesses and deaths, balancing her role as a daughter and mother. She finds solace in childhood memories and the legacy of her parents' love.
A House with Good Bones	Samantha, a 32-year-old archaeoentomologist, returns to her childhood home on Lammergeier Lane in North Carolina, where she confronts her family's dark past, including her grandmother Gran Mae's mysterious and malevolent legacy. As Samantha navigates her mother's strange behavior and the eerie presence of vultures, she uncovers secrets involving ritual magic, a jar of human teeth, and the supernatural ”underground children.” With the help of her friend Gail and handyman Phil, Samantha faces the haunting manifestations of her family's history. The novel explores themes of family, memory, and the supernatural, blending elements of horror and fantasy.	Samantha Montgomery returns home to find her mother acting strangely and the house devoid of insects. She uncovers a dark history involving her great-grandfather, a sorcerer, and her grandmother, who used roses to wield power. With help from Gail and Phil, she confronts the terrifying ”underground children,” using rose power to banish threats.	The protagonist returns to their grandmother's unchanged garden, filled with roses but mysteriously devoid of insects. They uncover unsettling truths about their grandmother's past and their mother's current state of mind. The narrative explores themes of family legacy and the passage of time.	Samantha, an archaeoentomologist, returns to her childhood home and finds herself investigating insect collections. Dealing with sleep paralysis and memories of her grandmother, she discovers the peculiar absence of insects in the garden. She navigates family dynamics and her mother's anxiety amid an eerie atmosphere.

F Prompts

F.1 Minion

RemoteLM

~~~
We need to answer the following question based on a {doc_type}.

### Question
{query}

### Instructions
You will not have direct access to the {doc_type}, but can chat with a small language
     model which has read the entire thing.

Feel free to think step-by-step, but eventually you must provide an output
in the format below:

<think step by step here>
```json
{{
    "message": "<your message to the small language model>"
}}
~~~
```

LocalLM

```
You will help a user answer the following question based on a {doc_type}.

Read the {doc_type} below and prepare to answer questions from an expert user.
### {doc_type}
{context}

### Question
{query}
```

Conversation

```
Here is the response from the small language model:

### Response
{response}

### Instructions
Analyze the response and think-step-by-step to determine if you have enough
information to answer the question.

If you have enough information, provide a final numeric answer in the format
below.

<think step by step here>
```json
{{
    "decision": "provide_final_answer",
    "answer": "<your answer>"
}}
```

Otherwise, request additional information from the small language model by
outputting the following:

<think step by step here>
```json
{{
    "decision": "request_additional_info",
    "message": "<your message to the small language model>"
}}
```
```

F.2 MINION$\mathcal{S}$

MINION$\mathcal{S}$: FINANCEBENCH

Decompose
```

```
Decomposition Round #{step_number}

You do not have access to the raw document(s), but instead can assign tasks to small
     and less capable language models that can read the document(s).
Note that the document(s) can be very long, so each task should be performed only over
     a small chunk of text.

Write a Python function that will output formatted tasks for a small language model.
Make sure that NONE of the tasks require calculations or complicated reasoning.
Any information you mentioned in your task should be given an extraction task.

Please use chunks of {pages_per_chunk} pages using the `chunk_on_multiple_pages(doc =
     context, pages_per_chunk ={pages_per_chunk})` function.

If you have multiple tasks, consider using nested for-loops to apply a set of tasks to
     a set of chunks. Though it's not required to have more than one task.

{ADVANCED_STEPS_INSTRUCTIONS}

Assume a Pydantic model called `JobManifest(BaseModel)` is already in global scope. For
      your reference, here is the model:
```
{manifest_source}
```
Assume a Pydantic model called `JobOutput(BaseModel)` is already in global scope. For
     your reference, here is the model:
```
{output_source}
```
DO NOT rewrite or import the model in your code.

The function signature will look like:
```
{signature_source}
```

You can assume you have access to the following chunking function(s). Do not
     reimplement the function, just use it.
```
{chunking_source}
```
```

Worker

```
Your job is to complete the following task using only the context
    below. The context is a chunk of text taken arbitrarily from a
    document, it might or might not contain relevant information to
    the task.

## Document
{context}

## Task
{task}

{advice}

Return your result in JSON with the following keys: "explanation",
    "citation", and "answer".

- "explanation": A concise statement of your reasoning or how you
    concluded your answer.
- "citation": A direct snippet of the text that supports your
    answer. If nothing is found, put "None".
- "answer": The extracted answer. If nothing is found, put "None".

Be certain to only rely on the provided text. If you cannot
    determine the information confidently from this chunk, respond
    with "None" for all fields.
```

Synthesize

~~~
Now synthesize the findings from multiple junior workers (LLMs).
Your task is to finalize an answer to the question below **if and
    only if** you have sufficient, reliable information.
Otherwise, you must request additional work.

---
## Inputs
1. Question to answer:
{question}

2. Collected Job Outputs (from junior models):
{extractions}

---
First think step-by-step and then answer the question using the
    exact format below.

## ANSWER GUIDELINES
1. **Determine if the collected Job Outputs provide enough
    trustworthy, consistent evidence to confidently answer the
    question.**
   - If the data is incomplete or contradictory, do NOT guess.
       Instead, specify what is missing.
   - If the evidence is sufficient, provide a final answer.

2. **Be conservative.** When in doubt, ask for more information.

3. **Address conflicts.** If multiple jobs give different answers,
    rely on whichever is best supported by a valid "explanation" and
     "citation".
   - If you need more information from the conflicting jobs you
       could request additional work from those specific jobs (be
       sure to mention the specific job IDs in your additional_info
       field).
   - Then, in the next round you can make a smaller set of jobs to
       determine which answer is correct.

4. **Required JSON Output**: You must output a JSON object with
    these keys:
   - "decision": Must be either "provide_final_answer" OR "
       request_additional_info".
     - Use "provide_final_answer" if you have enough information.
     - Use "request_additional_info" if you cannot conclusively
         answer.
   - "explanation": A short statement about how you arrived at your
        conclusion or what is still missing.
   - "answer": The final answer string if "decision"="
       provide_final_answer", or null otherwise. Should contain ONLY
        the final answer, without additional calculations or
       explanations.

Here is the template for your JSON response (with no extra text
    outside the JSON):

<think step-by-step here>
```json
{{
"decision": "...",
"explanation": "...",
"answer": "... or null", # Good answer format: "0.56"; Bad answer
    format: "The ratio is calculated as 1-0.27*2 = 0.56"
}}
~~~

Important:
- If there is not enough information, set "answer" to null, set "
    decision" to "request_additional_info", and specify exactly what
     else you need in "missing_info" and from which job IDs.
```
```

MINIONS: LONGHEALTH

Decompose

```
# Decomposition Round #{step_number}

You do not have access to the raw document(s), but instead can assign tasks to small and less capable
        language models that can read the document(s).
Note that the document(s) can be very long, so each task should be performed only over a small chunk
      of text.

Write a Python function that will output formatted tasks for a small language model.
Make sure that NONE of the tasks require multiple steps. Each task should be atomic!
Consider using nested for-loops to apply a set of tasks to a set of chunks.
The same `task_id` should be applied to multiple chunks. DO NOT instantiate a new `task_id` for each
      combination of task and chunk.
Use the conversational history to inform what chunking strategy has already been applied.

{ADVANCED_STEPS_INSTRUCTIONS}

Assume a Pydantic model called `JobManifest(BaseModel)` is already in global scope. For your
      reference, here is the model:
```
{manifest_source}
```
Assume a Pydantic model called `JobOutput(BaseModel)` is already in global scope. For your reference,
        here is the model:
```
{output_source}
```
DO NOT rewrite or import the model in your code.

The function signature will look like:
```
{signature_source}
```


You can assume you have access to the following chunking function(S). Do not reimplement the function
      , just use it.
```
{chunking_source}
```

Here is an example
```
task_id = 1 # Unique identifier for the task
for doc_id, document in enumerate(context):
    # if you need to chunk the document into sections
    chunks = chunk_by_section(document)
    # or if you need to chunk the document into pages
    chunks = chunk_by_page(document)

    for chunk_id, chunk in enumerate(chunks):
        # Create a task for extracting mentions of specific keywords
        task = (
            "Extract all mentions of the following keywords: "
            "'Ca19-9', 'tumor marker', 'September 2021', 'U/ml', 'Mrs. Anderson'."
)
        job_manifest = JobManifest(
            chunk_id=f"doc_id_chunk_id",
            task_id=task_id,
            chunk=chunk,
            task=task,
            advice="Focus on extracting the specific keywords related to Mrs. Anderson's tumor marker
                  levels."
)
        job_manifests.append(job_manifest)
```
```

$\mathbf{p}_{\text{worker}}$

```
Your job is to complete the following task using only the context below. The context is
     a chunk of text taken arbitrarily from a document, it might or might not contain
     relevant information to the task.

## Document
{context}

### Question you are trying to answer:
{question}

# You have been instructed to extract information pertaining to the following concepts:
# \"Date of visit\", {task}

Format your response as follows:
{{
"Date of visit" : "`direct quote extracted text`",
"<keyword_1>" : "`direct quote extracted text`",
"<keyword_2>" : "`direct quote extracted text`",
...
}}

Can you please extract the relevant sections from the document that are related to the
     concepts provided? Extract direct quotes or sentences. If concept is not mentioned
     , leave it out.

Your Answer:
```

$P_{\text{synthesize}}$

```
Answer the following by the synthesizing findings from multiple junior workers (LLMs).


---
## Inputs
1. Question to answer:
{question}

2. Collected Job Outputs (from junior models):
{extractions}

---
First think step-by-step and then answer the question using the exact format below.

## ANSWER GUIDELINES

**Required JSON Output**: You must output exactly one JSON object with these keys:
   - "decision": Must be  "provide_final_answer".
   - "explanation": A short statement about how you arrived at your conclusion or what
        is still missing.
   - "answer": The final answer string (that matches one of the provided options) if "
        decision"="provide_final_answer", or null otherwise.


Here is the template for your JSON response:

<think step-by-step here>


{{
"decision": "...",
"explanation": "...",
"answer": "...",
}}


Now, carefully inspect the question, think step-by-step and perform any calculations
     before outputting the JSON object. If answer choices are provided, your answer
     must **exactly** match one of the answer choices.

Question:
{question}

Your Answer:
```

MINIONS: QASPER

p$_{\text{decompose}}$

```
# Decomposition Round #{step_number}

You do not have access to the raw document(s), but instead can assign tasks to small
     and less capable language models that can read the document(s).
Note that the document(s) can be very long, so each task should be performed only over
     a small chunk of text.

Write a Python function that will output formatted tasks for a small language model.
Make sure that NONE of the tasks require multiple steps. Each task should be atomic!
Consider using nested for-loops to apply a set of tasks to a set of chunks.
The same `task_id` should be applied to multiple chunks. DO NOT instantiate a new `
     task_id` for each combination of task and chunk.
Use the conversational history to inform what chunking strategy has already been
     applied.

{ADVANCED_STEPS_INSTRUCTIONS}

Assume a Pydantic model called `JobManifest(BaseModel)` is already in global scope. For
      your reference, here is the model:
```
{manifest_source}
```
Assume a Pydantic model called `JobOutput(BaseModel)` is already in global scope. For
     your reference, here is the model:
```
{output_source}
```
DO NOT rewrite or import the model in your code.

The function signature will look like:
```
{signature_source}
```


You can assume you have access to the following chunking function(S). Do not
     reimplement the function, just use it.
```
{chunking_source}
```

Here is an example
```
task_id = 1 # Unique identifier for the task
for doc_id, document in enumerate(context):
    # if you need to chunk the document into sections
    chunks = chunk_by_section(document)
    # or if you need to chunk the document into pages
    chunks = chunk_by_page(document)

    for chunk_id, chunk in enumerate(chunks):
        # Create a task for extracting mentions of specific keywords
        task = (
            "Extract all mentions of the following keywords: "
            "'Ca19-9', 'tumor marker', 'September 2021', 'U/ml', 'Mrs. Anderson'."
)
        job_manifest = JobManifest(
            chunk_id=f"doc_id_chunk_id",
            task_id=task_id,
            chunk=chunk,
            task=task,
            advice="Focus on extracting the specific keywords related to Mrs. Anderson'
                s tumor marker levels."
)
        job_manifests.append(job_manifest)
```
```

p$_{\text{worker}}$

```
Your job is to complete the following task using only the context below. The context is
     a chunk of text taken arbitrarily from a document, it might or might not contain
     relevant information to the task.

## Document
{context}

### Question you are trying to answer:
{question}

# You have been instructed to extract information pertaining to the following concepts:
# \"Date of visit\", {task}

Format your response as follows:
{{
"Date of visit" : "`direct quote extracted text`",
"<keyword_1>" : "`direct quote extracted text`",
"<keyword_2>" : "`direct quote extracted text`",
...
}}

Can you please extract the relevant sections from the document that are related to the
     concepts provided? Extract direct quotes or sentences. If concept is not mentioned
     , leave it out.

Your Answer:
```

Psynthesize

```
Answer the following by the synthesizing findings from multiple junior workers (LLMs).


---
## Inputs
1. Question to answer:
{question}

2. Collected Job Outputs (from junior models):
{extractions}

---
First think step-by-step and then answer the question using the exact format below.

## ANSWER GUIDELINES

**Required JSON Output**: You must output exactly one JSON object with these keys:
   - "decision": Must be "provide_final_answer" or "need more information"
   - "explanation": A short statement about how you arrived at your conclusion or what
        is still missing.
   - "answer": a final answer that is a text span pulled directly from the job output
        citations.


Here is the template for your JSON response:

<think step-by-step here>

{{
"decision": "...",
"explanation": "...",
"answer": "..,",
}}

Now, carefully inspect the question, think step-by-step and perform any calculations
     before outputting the JSON object.
- If answer choices are provided, your answer must **exactly** match one of the answer
     choices.
- Don't paraphrase the final answer --- extract text directly from the document(s) or
     previous job outputs.

Question:
{question}

Your Answer:
```