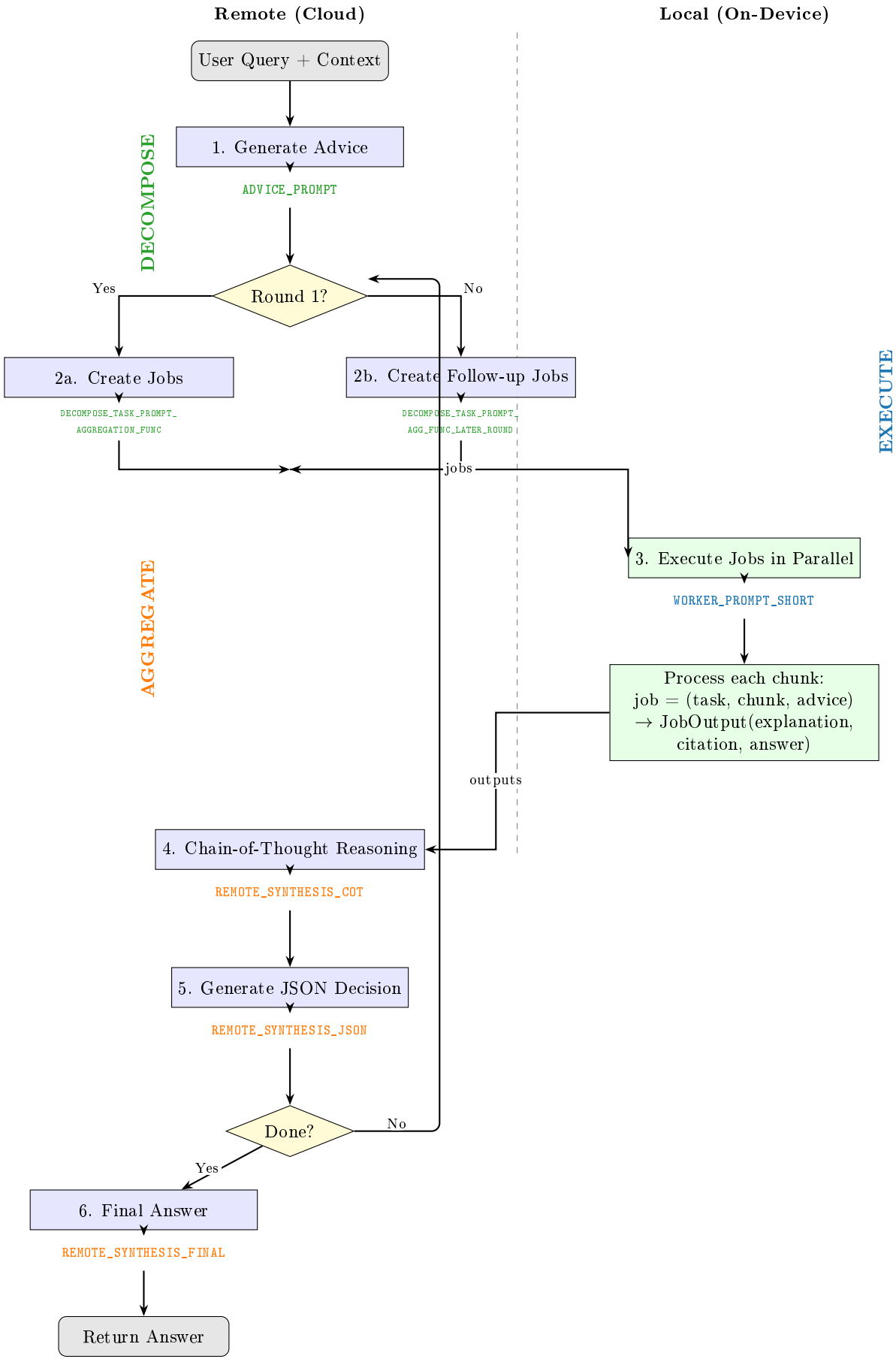
MinionS Prompt Reference

Source: `minions/prompts/minions.py`

Contents

1 MinionS Protocol Flow

The following diagram shows the complete flow of prompts through the MinionS protocol, including all three modules and their interactions.



Key:

- Blue boxes: Remote LLM operations (cloud)

- Green boxes: Local LLM operations (on-device)
- Green prompts: DECOMPOSE module
- Blue prompts: EXECUTE module
- Orange prompts: AGGREGATE module

2 Prompt-to-Module Mapping

Prompt Name	Module	When Used
ADVICE_PROMPT	DECOMPOSE	Initial guidance
DECOMPOSE_TASK_PROMPT_AGGREGATION_FUNC	DECOMPOSE	Round 1
DECOMPOSE_TASK_PROMPT_AGG_FUNC_LATER_ROUND	DECOMPOSE	Rounds 2+
WORKER_PROMPT_SHORT	EXECUTE	Local worker execution
REMOTE_SYNTHESIS_COT	AGGREGATE	Chain-of-thought
REMOTE_SYNTHESIS_JSON	AGGREGATE	Decision output
REMOTE_SYNTHESIS_FINAL	AGGREGATE	Force final answer

Note: Retrieval variants (DECOMPOSE_RETRIEVAL_TASK_PROMPT_*) exist but share the same structure with BM25 instructions embedded.

3 Prompt Definitions

ADVICE_PROMPT — Generates advice for workers on what information to extract.

ADVICE_PROMPT

```
We need to answer the following question based on {metadata}.:

## Question
{query}

---

Please provide succinct advice on the critical information we need to extract
from the {metadata} to answer this question.

Also consider the following constraints:
- In your response do NOT use numbered lists.
- Do NOT structure your response as a sequence of steps.
```

DECOMPOSE_TASK_PROMPT_AGGREGATION_FUNC — Main decomposition prompt for Round 1, requiring two Python functions: `prepare_jobs()` and `transform_outputs()`.

DECOMPOSE_TASK_PROMPT_AGGREGATION_FUNC

```
# Decomposition Round #{step_number}

You (the supervisor) cannot directly read the document(s). Instead, you can
assign small, isolated tasks to a less capable worker model that sees only
a single chunk of text at a time.

## Your Job: Write Two Python Functions

### FUNCTION #1: `prepare_jobs(context, prev_job_manifests, prev_job_outputs)`
- Break the document(s) into chunks
- Each job must be **atomic** and require only information from the **single chunk**
- If you need to repeat the same task on multiple chunks, **re-use** the same `task_id`
- Limit yourself to **up to {num_tasks_per_round} tasks** total

## BM25 Retrieval Instructions
- For each subtask you create, create keywords for retrieving relevant chunks.
  Extract precise keyword search queries that are **directly derived** from
  the user's question and the subtask.
- Assign high weights to the most essential terms (e.g. terms, dates, numerical
  values) to maximize retrieval accuracy.
- Choose a higher value for `k` (15) if you are unconfident about your keywords.

### FUNCTION #2: `transform_outputs(jobs) -> str`
- Accepts the worker outputs for the tasks you assigned
```

```
- Apply any **filtering logic** (e.g., drop irrelevant or empty results)
- **Aggregate outputs** by `task_id` and `chunk_id`
- Return one **aggregated string** suitable for further supervisor inspection

## Important Reminders:
- **DO NOT** assign tasks that require reading multiple chunks
- Keep tasks **chunk-local and atomic**
- **You** (the supervisor) are responsible for aggregating outputs
```

DECOMPOSE_TASK_PROMPT_AGG_FUNC_LATER_ROUND — Decomposition prompt for Rounds 2+, with access to previous job outputs.

DECOMPOSE_TASK_PROMPT_AGG_FUNC_LATER_ROUND

```
# 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.

# Your job is to write two Python functions:

Function #1 (prepare_jobs): 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.
-> Use the conversational history to inform what chunking strategy has been applied.
-> You are provided access to the outputs of the previous jobs (see prev_job_outputs).
-> If helpful, you can reason over the prev_job_outputs vs. the original context.
-> If tasks should be done sequentially, wait for the next round.

## BM25 Retrieval Instructions
- For each subtask you create, create keywords for retrieving relevant chunks.
  Extract precise keyword search queries that are **directly derived** from
  the user's question and the subtask.
- Assign high weights to the most essential terms (e.g. terms, dates, numerical
  values) to maximize retrieval accuracy.
- Choose a higher value for `k` (15) if you are unconfident about your keywords.

Function #2 (transform_outputs): aggregate outputs for supervisor review.
-> Filter jobs based on output (write a custom filter function).
-> Aggregate the jobs based on the task_id and chunk_id.

## Using Previous Round Information
Our conversation history includes information about previous rounds of jobs
and their outputs. Use this information to inform your new jobs:
- Based on the Job outputs above, subselect `chunk_id`s that require further
  reasoning and are relevant to the question
- Reformat tasks that are not yet complete.
- Make your `advice` more concrete.
```

WORKER_PROMPT_SHORT — Local worker prompt using Pydantic model specification.

WORKER_PROMPT_SHORT

```
Here is a document excerpt:

{context}

---------------------------------
And here is your task:

{task}

---------------------------------
And here is additional higher-level advice on how to approach the task:

{advice}

---------------------------------

Your response should be a `JobOutput` object:
```python
class JobOutput(BaseModel):
  explanation: str  # A concise statement of your reasoning
  citation: str | None  # A direct snippet supporting your answer
  answer: str | None  # Your answer to the question
```
Your response:
```

REMOTE_SYNTHESIS_COT — Chain-of-thought synthesis prompt for step-by-step reasoning.

REMOTE_SYNTHESIS_COT

```
Now synthesize the findings from multiple junior workers (LLMs).
Your task is to analyze the collected information and think step-by-step
about whether we can answer the question. Be brief and concise.

## Previous Progress
{scratchpad}

## Inputs
1. Question to answer: {question}
2. Collected Job Outputs (from junior models): {extractions}

## Instructions
Think step-by-step about:
1. What information we have gathered
2. Whether it is sufficient to answer the question
3. If not sufficient, what specific information is missing
4. If sufficient, how we would calculate or derive the answer
5. If there are conflicting answers: Use citations to select the correct one

Be brief and concise. No need for structured output.
```

REMOTE_SYNTHESIS_JSON — JSON output format specification for synthesis decisions.

REMOTE_SYNTHESIS_JSON

```
Based on your analysis, return a single JSON object:

{
  "explanation": "",
  "feedback": null,
  "decision": "",
  "answer": null,
  "scratchpad": ""
}

Field Descriptions:
- explanation: A brief statement of your reasoning
- feedback: Specific information to look for, if needed (null if not applicable)
- decision: Either "provide_final_answer" or "request_additional_info"
- answer: The final answer if providing one; null otherwise
- scratchpad: Summary of gathered information for future reference
```

REMOTE_SYNTHESIS_FINAL — Final answer synthesis prompt (forces a final answer).

REMOTE_SYNTHESIS_FINAL

```
Now provide the final answer based on all gathered information.

## Previous Progress
{scratchpad}

## Inputs
1. Question to answer: {question}
2. Collected Job Outputs (from junior models): {extractions}

Return a single JSON object:
{
  "explanation": "",
  "feedback": null,
  "decision": "",
  "answer": null,
  "scratchpad": ""
}

Field Descriptions:
- explanation: Brief statement of your reasoning
- feedback: Any specific information lacking (NO CODE). Use null if not needed
- decision: must be "provide_final_answer"
- answer: Final answer
- scratchpad: Summary of gathered information
```