Context Engineering and Memory: Takeaways

by Dataquest Labs, Inc.

Syntax

- Truncating a tool result to a character limit:

```
def truncate(text, limit=MAX_TOOL_OUTPUT_CHARS):
    if len(text) <= limit:
        return text
    cut = len(text) - limit
    return text[:limit] + f"\n[truncated {cut} characters. Print less, e.g. a summary or the
first rows.]"
```

- Building a truncated tool-result message:

```
def to_tool_message(tool_call_id, result):
    return {"role": "tool", "tool_call_id": tool_call_id, "content":
truncate(json.dumps(result))}
```

- Loading a notes file from the workspace:

```
def load_notes(workspace):
    path = Path(workspace) / NOTES_FILE
    if path.exists():
        return path.read_text(encoding="utf-8")
    return None
```

- Adding earlier notes to the system prompt:

```
def build_system_prompt(workspace):
    notes = memory.load_notes(workspace)
    if notes:
        return SYSTEM_PROMPT + f"\nYour notes from earlier runs ({memory.NOTES_FILE}):
\n\n{notes}\n"
    return SYSTEM_PROMPT
```

- Finding a safe split point before an assistant message:

```
def split_point(messages):
    index = max(2, len(messages) - KEEP_RECENT_MESSAGES)
    while index > 2 and as_dict(messages[index])["role"] != "assistant":
        index -= 1
    return index
```

- Replacing the older messages with a summary:

```
start = split_point(messages)
older, recent = messages[2:start], messages[start:]
return messages[:2] + [summary] + recent, usage
```

- Compacting once a call's input passes the threshold:

```
if usage.prompt_tokens > config.compact_at_tokens:
    messages, summary_usage = memory.compact(messages, client, config.model,
config.reasoning_effort)
    input_tokens += summary_usage["input_tokens"]
    output_tokens += summary_usage["output_tokens"]
```

Concepts

- **Context**: the running list of messages the agent sends to the model on every step. Each call re-sends the whole list, so it grows with every step and every tool result.
- An agent's context grows faster than a chatbot's, because it reads data into tool results, it takes many steps for one question, and no person sits between the steps to keep the turns short.
- **Truncation**: capping a tool result at a character limit before it enters the messages, with a short note in its place so the model knows the output was shortened and can ask for less.
- A result is large when it is too big to put in front of the model, measured against the context window rather than disk or memory. What fills the context is the size of a tool's output, not the size of the file it read.
- **Notes file**: a short NOTES.md the agent keeps in its workspace and updates with reusable facts. Because it lives in the workspace and not in the conversation, it survives into later runs and loads into the system prompt.
- **Compaction**: replacing an older stretch of the conversation with a model-written summary during a run, while keeping the system prompt, the task, and the most recent messages.
- Dropping the oldest messages breaks an agent. Index 1 holds the task, and removing a message from the middle can leave a tool result with no matching call, which the API rejects.
- The compaction trigger reads the token count the model already reports in response.usage.prompt_tokens instead of counting tokens by hand, and compacts once a call's input passes a threshold.

Resources

- pathlib filesystem paths (Python documentation)
- json encoder and decoder (Python documentation)
- tiktoken, a fast BPE tokenizer (OpenAI GitHub)

© Dataquest Labs, Inc. All rights reserved.