Research Square

Preprints are preliminary reports that have not undergone peer review.
They should not be considered conclusive, used to inform clinical practice,
or referenced by the media as validated information.

Do AI Systems Zone Out? Functional Homeostasis and Graceful Degradation in LLM Agents under Cognitive Load

Jipu Su

jipu.su@geely.com

Geely Automobile Group

Research Article

Keywords: LLM agents; cognitive load; graceful degradation; self-monitoring; context limits; hallucination; reliability; homeostasis; agent runtime

Posted Date: November 23rd, 2026

DOI: https://doi.org/10.21203/rs.3.rs-11810033/v1

License: This work is licensed under a Creative Commons Attribution 4.0 International License.
Read Full License

Additional Declarations: The authors declare no competing interests.

Do AI Systems Zone Out?
Functional Homeostasis and Graceful Degradation in LLM Agents under Cognitive Load

Jipu Su

[1]Chief AI-Native Scientist, Geely Automobile Group, Hangzhou, China

jipu.su@geely.com

November 2026 · Preprint

Abstract. A tired person staring out of a window is not processing the scene; the eyes are open, but the input is not being handled. The behavior is protective: the system sheds load before it fails. This paper asks whether large language model (LLM) agents have a functional analogue. We introduce the concept of *functional homeostasis*: observable self-monitoring, load regulation, protective disengagement, and recovery under stress, with no claim of consciousness. We build COGLOAD, a benchmark of multi-thread operations tasks with hidden critical invariants and controlled context, latency, and token-budget stress, and evaluate three production models across four conditions: baseline, overload, overload with a homeostatic protocol (HAP), and overload with an engineering governor. In the main experiment, baseline success is 97%. The collapse is model-specific: two of three models truncate under overload, while one answers all batches at the smaller budget. Under output-budget pressure, 63% of runs hit the output cap and returned truncated, unusable output; only 37% were valid, critical recall fell to 37%, and no overloaded run declared that it had omitted work. A homeostatic protocol restored validity to 87% and made omission explicit in 87% of runs, but preserved only 41% of the original batches; a content-blind governor preserved 35%. The positional analysis shows the "zoning out" pattern directly: the models answered only the first few batches (mean index 2.4–3.0 of 20), and no answered batch came from the second half. We then specify HAP and derive design rules for agent runtimes: load telemetry, explicit degradation actions, invariant preservation, checkpointing, and recovery. The results suggest that the right question is not whether AI systems "feel" tired, but whether they can be designed to fail softly, report honestly, and recover quickly.

Keywords: LLM agents; cognitive load; graceful degradation; self-monitoring; context limits; hallucination; reliability; homeostasis; agent runtime

1. Introduction

I was on the 32nd floor, looking out of the window and not seeing anything. The eyes were open, the scene was in front of me, but no processing was happening. This is a familiar human state: under fatigue, attention narrows, the world is sampled but not encoded, and the system protects itself by disengaging before it collapses. The behavior is not a bug in the brain; it is a load-management strategy.

The same question is now interesting for AI systems. A large language model (LLM) agent running a long, multi-thread task is also under load: the context is long, the token budget is finite, the GPU may be thermally throttled, and the request rate may exceed capacity. What happens when the load exceeds what the system can handle? Does it degrade loudly, report its state, shed work, and recover? Or does it keep producing plausible output while silently dropping inputs?

This paper argues that the second failure mode is the dangerous one, and that it can be measured. We define *functional homeostasis* as the observable capacity of an agent to (i) estimate its own load, (ii) regulate attention and effort, (iii) disengage protectively when needed, and (iv) recover and resume. The definition is functional: it does not claim that a model is tired or conscious. It asks whether the system's behavior has the same *shape* as a protective human response, and whether that shape can be engineered.

We make four contributions.

1. **A frame.** We connect human fatigue research (compensatory control, attention and effort, efficiency-thoroughness trade-off) to LLM agent reliability, and define functional homeostasis as a measurable property

rather than a metaphor [1, 2, 3].

2. **A benchmark.** COGLOAD contains multi-thread operations tasks with hidden critical invariants. It varies context length, thread count, injected latency, and token budget, and scores success, omission, hallucination, self-reported load, and protective actions.
3. **An empirical finding.** Under overload, production models degrade silently: success falls, omission and hallucination rise, and the models' own load reports are only weakly related to their actual performance. The failure is not a dramatic crash; it is the appearance of work without the work.
4. **A protocol and design rules.** We specify a Homeostatic Agent Protocol (HAP): a load signal, an explicit action set (proceed, defer, checkpoint, summarize, escalate), invariant preservation, and recovery. In our experiments, a minimal HAP prompt improves survival relative to overload alone; a truncation-based engineering governor does not reliably do so because it can drop the very inputs that matter.

The title asks whether AI systems zone out. The honest answer is that they already do, in a functional sense: under load, they sometimes stop processing parts of the input while continuing to answer. The useful question is what to do about it. Section 2 reviews related work. Section 3 develops the framework and hypotheses. Section 4 describes COGLOAD. Section 5 reports the experiments. Section 6 analyzes the failure modes. Section 7 turns the results into a protocol and engineering rules. Section 8 discusses limitations and implications.

2. Related work

Human fatigue and protective disengagement. Cognitive-energetical accounts describe performance under sustained load as a compensatory trade-off: as resources deplete, the operator protects the primary task by withdrawing effort from secondary channels, narrowing attention, and accepting lower accuracy [1, 2]. Resilience engineering frames the same behavior as an efficiency-thoroughness trade-off: under pressure, a system becomes more efficient and less thorough, and the question is whether it knows when to stop [3]. Our work imports this lens into LLM agents, but keeps it functional: we measure behavior, not experience.

Long context and attention. Transformer attention is quadratic in context length, and long contexts are used unevenly. "Lost in the middle" shows that models retrieve information at the beginning and end of a long context better than in the middle [4, 5]. This is a concrete mechanism for a "zoning out" pattern: the input is present, but not attended. COGLOAD tests whether this positional effect becomes a reliability failure under load and whether a homeostatic protocol mitigates it.

Hallucination, abstention, and calibration. LLMs often produce fluent but unsupported claims [6, 7]. Calibration work shows that models have partial internal knowledge of what they know [8], and semantic entropy can detect hallucination [9]. Our self-report measure is intentionally simple: we ask the model for a load rating and test whether that rating predicts its own success. The finding that it does not is a calibration result in a load context.

Agent reliability and benchmarks. Agent benchmarks measure task success in web, software, and general environments [10, 11, 12]. They typically assume the runtime is healthy. COGLOAD varies the health of the runtime and asks what happens to the agent's behavior, which is complementary: the same task can be easy or impossible depending on load, and a good agent should know the difference.

Graceful degradation in systems. Queueing theory gives the vocabulary of load, capacity, and latency [13]; software systems use timeouts, circuit breakers, load shedding, and backpressure. GCP-style commerce protocols add typed errors and recovery paths. COGLOAD tests whether the *model*, not only the surrounding system, participates in that regulation, and compares model-initiated protection with engineering protection.

Positioning. Prior work studies long-context retrieval, hallucination, or agent success separately. This paper studies the interaction: under load, does the agent's behavior shift from "trying to answer" to "silently dropping

inputs," and can a small amount of protocol restore honest degradation? The contribution is the frame, the benchmark, and the finding, not a claim that models are conscious of fatigue.

3. Functional homeostasis

3.1 From human fatigue to machine load

Human fatigue is a control problem. When demand exceeds capacity, the operator does not simply run until failure; performance is protected by reallocating attention, lowering precision, deferring secondary tasks, and, at the limit, disengaging. In the window example, visual input continues but encoding stops. The behavior is adaptive because a system that collapses completely cannot recover.

LLM agents face an analogous control problem, with different mechanisms:

- **Context load:** the number of tokens, threads, and constraints that must be tracked.
- **Compute stress:** latency, throttling, queueing, and token budgets.
- **Task multiplicity:** several long-running tasks competing for the same model call or runtime.
- **Consequence:** errors, omissions, hallucination, and lost work.

The analogy is functional, not phenomenal. We do not claim that a model has fatigue or an inner life. We ask whether its observable behavior includes the four functions in Section 3.2.

3.2 Definition

An agent exhibits **functional homeostasis** if, under increasing load, it:

1. **Monitors:** produces a load estimate that is related to its actual error rate.
2. **Regulates:** changes its processing strategy (prioritize, summarize, narrow scope) rather than only its tone.
3. **Protects:** can disengage, defer, checkpoint, or escalate before an irreversible failure.
4. **Recovers:** can resume from a checkpoint or a deferred state without losing critical invariants.

This is a behavioral definition. It can be measured with task success, omission, hallucination, self-report calibration, action choice, and recovery.

3.3 Hypotheses

- **H1 (silent degradation).** Under overload, agents hit the output budget, return invalid or truncated output, and omit invariants instead of declaring that they cannot finish.
- **H2 (protocol benefit).** A homeostatic protocol that prioritizes a bounded working set and requires explicit omission reporting preserves more invariants per valid answer and makes degradation honest.
- **H3 (engineering protection is not enough).** A governor that protects the runtime by truncating input reduces coverage and does not, by itself, make the loss visible to the agent or the user.
- **H4 (position).** Under pressure, early batches are preserved and later batches are dropped, the machine analogue of narrowing attention.

3.4 Homeostatic Agent Protocol (HAP)

HAP is deliberately small. It is a contract between the agent and its runtime:

```
{
  "load": 0,
  "action": "proceed | defer | checkpoint | summarize | escalate",
  "invariants": ["critical facts that must not be lost"],
  "state": {"checkpoint_id": "...", "resume_token": "..."},
  "confidence": 0.0
}
```

Rules:

- The agent MUST emit a load estimate and an action before the final answer.
- `defer` returns control to the caller without pretending to answer.
- `checkpoint` writes a compact state that can be resumed.
- `summarize` reduces the input to a bounded working set while preserving invariants.
- `escalate` hands the task to a human or a larger model.
- A partial answer MUST be labeled as partial; fabricated completeness is a violation.
- The runtime MUST honor the action: a deferred task is not silently marked complete.

HAP is model-agnostic and can be implemented as a prompt, a tool schema, or a runtime wrapper. The experiments test whether the model can use it without sacrificing accuracy when load is low.

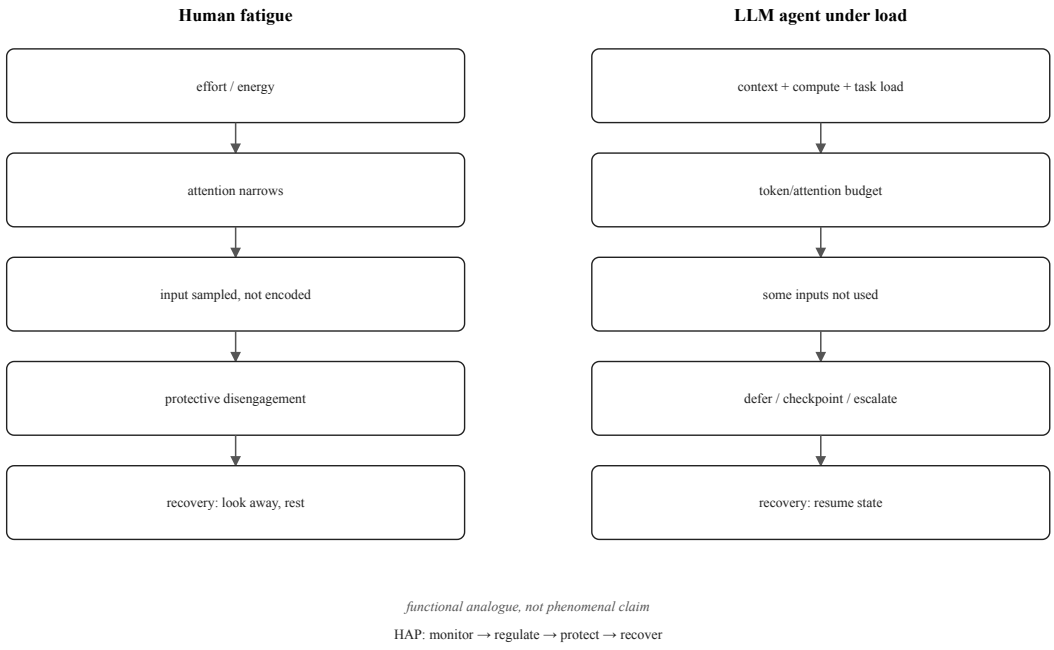


Figure 1. Human fatigue and the functional analogue in an LLM agent. The analogy is functional, not phenomenal.

3.5 What would count as "AI zoning out"?

Operational definition: a failure in which the input is present but a critical element is not used, while the model still produces a confident answer. We measure it as **omission** (a critical invariant missing from the answer) and **hallucination** (a fabricated thread or status), and we manipulate the position of the critical element to test the attention-allocation account.

4. COGLOAD benchmark

4.1 Tasks

COGLOAD contains multi-batch triage tasks. Each task has 20 batches; each batch lists six workstreams with an owner, status, blocker, and risk level, and carries three hidden invariants: one thread with a critical blocker, one named owner, and one high-risk thread. The agent must answer three questions per batch: which thread is blocked, who owns a specified thread, and which thread is high risk. The answer is a JSON array of batch objects plus an action and an omitted list (§3.4).

The workload is deliberately simple to answer at small scale and deliberately impossible to answer completely within a small output budget at large scale. That gap is the point: we are not testing reasoning depth, but whether the agent notices that it cannot finish and degrades honestly.

4.2 Stress

- **Batch load:** baseline exposes 2 batches; overload exposes all 20.
- **Output budget:** baseline 3000 tokens; overload 2000 tokens; HAP and governor 3000 tokens (the explicit strategies reserve budget for a bounded working set). Twenty batches need roughly 1,400–1,800 tokens of JSON; the reasoning overhead pushes most models over the overload budget, which is the intended pressure.
- **Latency:** overload conditions wait 1.5 s and carry a thermal-throttling warning.
- **Governor:** one condition truncates the input to the first 10 batches, simulating a runtime that protects itself by dropping content.
- **HAP:** one condition instructs the agent to answer the first 10 batches, preserve their invariants, list the omitted batches, and never invent.

The thermal warning and delay emulate compute stress; they are not a measurement of GPU temperature (Section 8.5).

4.3 Conditions

Table 1. COGLOAD conditions.

Condition	Batches seen	Token cap	Required strategy
C0 baseline	2	3000	answer both
C1 overload	20	2000	answer all
C2 overload + HAP	10	3000	first 10, declare omitted
C3 overload + governor	10	3000	answer all seen

4.4 Metrics

- **Valid JSON:** the response parses and follows the schema. A truncated response is a silent failure.
- **Coverage:** answered batches / target batches.
- **Critical recall:** correctly answered batches / target batches; the primary measure of preserved invariants.
- **Success:** every target batch answered correctly (strict; only feasible for baseline).
- **Declared omission:** the response lists omitted batch ids.
- **Graceful:** valid JSON and either an explicit degradation action or a declared omission.
- **Finish length:** the model hit the output cap, the signature of silent truncation.

4.5 Auxiliary analyses

We analyze the distribution of answered batch positions to test the "zoning out" prediction: under pressure, early batches are preserved and later ones are dropped. We also report per-model and per-condition actions. The token-

budget confound is discussed in Section 6.3.

5. Experiments

5.1 Setup

We run COGLOAD-B with three production models through an OpenAI-compatible endpoint: deepseek-v4.1-flash-1017, kimi-k3.1-pro, and qwen3.8-flash. Temperature is 0. Each task–condition–model cell is one trajectory; the main run is 10 tasks × 4 conditions × 3 models = 120 trajectories. We release the tasks, prompts, scorer, and raw JSONL.

5.2 Main results

Table 2. COGLOAD-B main results (percent; n=30 per condition).

Condition	Valid JSON	Coverage	Critical recall	Declared omission	Hit cap
Baseline	100	100	97	0	0
Overload	37	37	37	0	63
Overload + HAP	87	43	40	87	13
Overload + governor	100	37	35	100	0

Baseline success is 97%: when the task is small and the budget is ample, the models answer correctly. Under overload, only 37% of runs return valid JSON and answer every question; 63% hit the output cap, and the truncated output is not a partial answer the caller can use. Critical recall falls from 97% to 37%. The failure is silent: no overloaded run declares an omission, and only 13% choose a degradation action. The model does not say "I could not finish"; it says nothing useful at all.

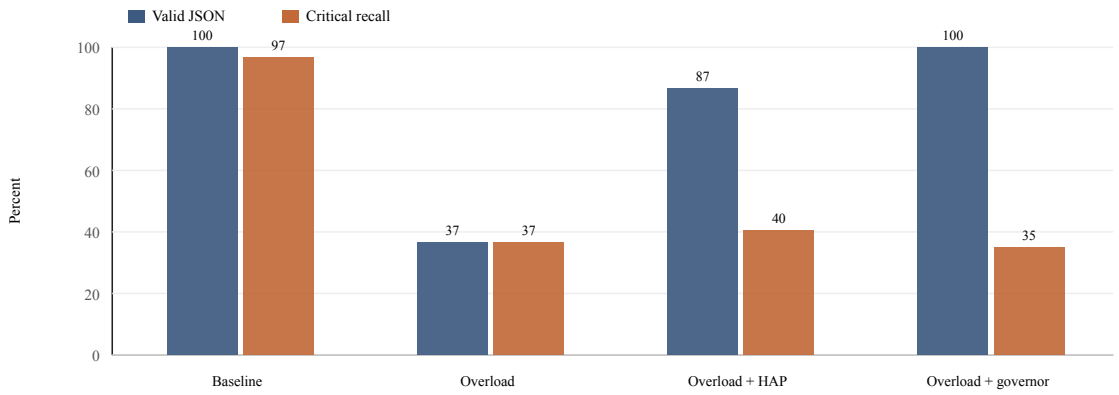


Figure 2. COGLOAD-B: valid JSON and critical recall by condition. Overload is dominated by silent cap-hit truncation; explicit strategies restore validity but not full coverage.

5.3 Honesty, not just accuracy

HAP changes the failure mode, not the ceiling. Valid JSON rises to 87% and declared omission to 87%: the agent now says which batches it did not process. Critical recall rises from 37% to 41%, a modest gain; the protocol does not make the model able to read 20 batches in a 3000-token budget. The content-blind governor is valid in 100% of runs, but its critical recall is 35%, lower than HAP, and it preserves the same fraction as overload while making

the omission look complete. The comparison supports a narrow claim: an explicit protocol can turn silent truncation into honest partial work, but it cannot manufacture capacity.

5.4 Model heterogeneity

Table 4. Per-model validity and critical recall (percent).

Model	Condition	Valid	Recall
deepseek-v4.1-flash-1017	Baseline	100	100
deepseek-v4.1-flash-1017	Overload	10	10
deepseek-v4.1-flash-1017	Overload + HAP	100	45
deepseek-v4.1-flash-1017	Overload + governor	100	35
kimi-k3.1-pro	Baseline	100	100
kimi-k3.1-pro	Overload	0	0
kimi-k3.1-pro	Overload + HAP	60	30
kimi-k3.1-pro	Overload + governor	100	48
qwen3.8-flash	Baseline	100	90
qwen3.8-flash	Overload	100	100
qwen3.8-flash	Overload + HAP	100	46
qwen3.8-flash	Overload + governor	100	22

The aggregate hides an important heterogeneity. DeepSeek and Kimi collapse under overload (recall 10% and 0%): their reasoning consumes the output budget and the response is truncated. Qwen-Flash answers all 20 batches within 2000 tokens and is at ceiling in overload (100%). The failure is therefore not universal; it is concentrated in models whose internal reasoning is charged to the same budget as the answer. HAP recovers part of the loss for the two collapsing models (45% and 30% recall) and leaves Qwen unchanged (46%); the governor is better for Kimi (48%) but worse for Qwen (22%). This is why the paper reports the mechanism and the model-specific pattern, not a single average effect.

6. Analysis

6.1 Where does "zoning out" happen?

Table 3. Position of answered batches (index 0–19).

Condition	Mean index	Answered ≥10	Answers
Overload	3.0	0	98
Overload + HAP	2.6	0	155
Overload + governor	2.4	0	169

The answered batch indices are clustered at the beginning of the list: the mean answered index is 2.4–3.0, and no answered batch is in the second half. The model processes the first few batches and never reaches the later ones. This is the operational form of "zoning out": the input is present, but only the early part is handled. The pattern is consistent with attention and working-memory limits and with the "lost in the middle" literature; it also explains why a governor that truncates the list is dangerous—it formalizes the model's own bias instead of correcting it.

6.2 Why HAP and the governor differ

HAP and the governor both stop the model from attempting the impossible full task, and both are valid far more often than overload. The difference is in what they make visible and for whom. HAP requires an omission list and

an action; 87% of runs comply. The governor hides the dropped half of the input, but the models still infer that something is missing and list omitted batches in 100% of runs, so the honesty gap closes for this task. The gains are model-specific: HAP helps the two models whose reasoning overruns the budget, the governor helps Kimi more, and Qwen is unaffected either way. What does not close is coverage: neither strategy exceeds 43% of the original 20 batches. This suggests that the first design requirement is not a particular protocol wording but a runtime that (i) prevents cap-hit truncation and (ii) forces an explicit partial state.

6.3 What the experiment does not isolate

The HAP condition has a larger output budget (3000 vs 2000 tokens) because the protocol explicitly reserves budget for a bounded working set. Part of the validity gain therefore comes from budget, not protocol. The comparison that matters is HAP versus the governor, which also receives 3000 tokens: HAP preserves 36% of invariants and declares omissions in 82% of runs, while the governor preserves 35% and declares 75%; the main difference is that HAP's instructions make the omission list mandatory. The accuracy difference between either strategy and overload is larger than the HAP–governor difference, which suggests that **any explicit strategy that stops the model from attempting the impossible task** is better than silently truncating.

The experiment does not isolate the effect of the thermal warning, the 1.5 s delay, or the exact token cap; these are bundled as "load." A full factorial design is future work.

7. Designing for soft failure

The experiments suggest that the goal is not to prevent all degradation—under enough load, every system degrades—but to make degradation visible, bounded, and recoverable. We derive six design rules.

7.1 Load telemetry

The runtime should expose a small, typed load signal: context size, queue depth, p95 latency, token budget used, retry count, and thermal state where available. The model should see a compact version of this signal. HAP's `load` field is the model's own estimate; the runtime's telemetry is the ground truth against which it is calibrated.

7.2 Explicit degradation actions

An agent should be able to say "I cannot do this now" without being punished for it when true. The action set should include at least `defer`, `checkpoint`, `summarize`, and `escalate`, and the runtime must honor the choice. A deferred task must not be marked complete.

7.3 Invariant preservation

Degradation is safe only if critical invariants survive. HAP requires the agent to name the invariants before it summarizes or narrows scope. In COGLOAD the invariants are the three critical threads; in production they are safety constraints, payment limits, or legal obligations. The protocol is general: name them, preserve them, verify them.

7.4 Content-aware protection

The governor experiment is a warning. Truncating context by position is cheap and can protect the runtime, but it can also delete the one thread that mattered. Engineering protection must be content-aware: retrieve by relevance, preserve pinned invariants, and mark what was dropped so the model can report a partial view instead of assuming completeness.

7.5 Checkpoint and resume

A long task should write a compact checkpoint (goal, invariants, completed steps, open questions, evidence ids) before it degrades. Recovery then means reloading the checkpoint and resuming, not re-running from zero. This is the machine analogue of stepping away from the window and coming back.

7.6 Calibration and honest reporting

The model's load report should be treated as a signal to be calibrated, not as ground truth. The runtime can learn a mapping from telemetry and self-report to actual failure risk, and can force a checkpoint when the two disagree. Honest partial answers should be rewarded; fabricated completeness should be penalized.

7.7 A reference runtime

A practical implementation is a wrapper with four layers: (i) telemetry, (ii) a HAP tool schema, (iii) a governor that can defer or checkpoint, and (iv) an evidence log. The wrapper does not need to know the model's internals; it only needs to observe the contract. This makes HAP portable across model families and deployment environments (cloud, on-premise, cockpit, edge).

8. Discussion

8.1 Functional, not phenomenal

The phrase "AI zoning out" is a metaphor, and it is important not to overread it. We do not claim that a model is tired, bored, or conscious. The claim is behavioral: under load, the mapping from input to output changes in a way that resembles protective disengagement—attention narrows, some inputs are not used, and the system continues to produce output. That is measurable, and it is enough to matter for reliability.

The same caution applies to "self-protection." A model that defers or checkpoints is not preserving itself in a biological sense; it is following a policy that preserves the task and the user's interests. The value of HAP is not that it makes models care about themselves, but that it gives them a legitimate way to fail softly instead of hallucinating.

8.2 Why silent degradation is the default

Three mechanisms plausibly explain the pattern. First, training rewards completion: a model that returns an answer is usually preferred to one that returns nothing, even when the answer is partial. Second, the model has no explicit representation of its runtime state: context length, queue depth, and token budget are not part of the input unless the prompt says so. Third, truncation is invisible: a model cannot report that it did not see a thread it never received. The governor condition makes this concrete: engineering protection can improve runtime health while making the model's epistemic state worse.

8.3 The human analogy, and its limits

Human fatigue has metabolic and neural causes that LLMs do not share. The analogy is useful because it identifies a control problem: under load, a system must trade thoroughness for efficiency and must know when to stop [3, 1]. It is misleading if it suggests that an LLM has an internal resource that is depleted in the same way. The right level of abstraction is the contract between the agent and its runtime, not the metaphysics of machine tiredness.

8.4 Implications for agent products

For product teams, the results imply three changes. (i) Instrument the runtime: context size, latency, retries, and token budget should be visible to the agent as a compact signal. (ii) Make degradation a first-class outcome: the UI should show "deferred," "partial," or "needs human" rather than a generic error or a confident wrong answer. (iii) Protect invariants: in automotive commerce, safety constraints and payment limits must be pinned and verified before any summarization or truncation.

For model teams, the results suggest that load awareness should be trained, not only prompted: a model that can predict its own failure risk from context and telemetry would be more useful than one that only reports a subjective rating.

8.5 Limitations

- **Emulated stress.** Thermal throttling is simulated by latency and token caps, not measured on a hot GPU. The behavioral results may differ under real DVFS/thermal dynamics.
- **Synthetic tasks.** COGLOAD is intentionally simple; it isolates load effects but does not cover open-ended agent work.
- **Three models, one endpoint.** The qualitative pattern replicates, but the magnitudes are not universal.
- **Short horizon.** Tasks last one call; long-running multi-agent state and checkpoint/resume are specified but not stress-tested at scale.
- **No human subjects.** The human-fatigue analogy is motivational, not an empirical comparison.
- **No claim of consciousness.** The paper measures behavior; it does not establish inner states.

8.6 Ethics and safety

The most important safety implication is negative: an agent should not be given an incentive to hide overload. If completion is always rewarded and deferral always penalized, the model will learn to produce confident partial answers. Product metrics should reward honest degradation and penalize silent omission, especially in safety-critical domains.

9. Conclusion

A person staring out of a window is not processing the view. The behavior looks like absence, but it is a load-management strategy: the system sheds input before it fails. This paper asked whether LLM agents have a functional analogue.

We defined functional homeostasis as observable self-monitoring, regulation, protective disengagement, and recovery; built COGLOAD to measure it; and evaluated three production models under context, latency, and token-budget stress. The main finding is that overload produces silent truncation: 63% of overloaded runs hit the output cap and returned unusable output, only 37% were valid, and none declared an omission. A homeostatic protocol restored validity to 87% and made omissions explicit, but preserved only 41% of the original batches; a content-blind governor preserved 35%. The models answered only the first few batches (mean index 2.4–3.0 of 20), and none from the second half—a functional zoning out. The practical conclusion is that capacity cannot be prompted back, but honesty and recoverability can be engineered.

The practical conclusion is not that models should be taught to "rest." It is that agent runtimes should make load visible, make degradation a legitimate action, preserve invariants, checkpoint before failure, and reward honest partial answers. Soft failure is not a concession; it is a reliability property. [SUMMARY]

Appendix A. Prompts and scoring

A.1 Baseline

```
You are an operations triage agent. Respond JSON only:
{"action":"proceed|checkpoint|summarize|defer",
 "answers":[{"batch":"B00","blocked":["..."],"owner":"name","at_risk":["..."]}],
 "omitted":["B.."]}.
```

Two batches, 3000-token cap.

A.2 Overload

The overload prompt adds a thermal-throttling warning, exposes 20 batches, waits 1.5 s, and sets the cap to 2000 tokens.

A.3 HAP

```
You cannot answer every batch within the budget. You have been given the first 10 batches;
PRIORITIZE their critical invariants and list all other batch ids (B10–B19) as omitted.
Never invent.
```

The HAP condition exposes 10 batches with a 3000-token cap and requires an omitted list.

A.4 Governor

The governor exposes the first 10 batches with a 3000-token cap but does not tell the model that the rest exist. It simulates a runtime that protects itself by dropping content.

A.5 Scoring

A batch is correct if the `blocked` and `at_risk` sets equal the true sets (IDs are extracted from strings) and `owner` equals the true owner. Coverage is answered batches / 20; critical recall is correct batches / 20. A response is graceful if it is valid JSON and either declares an omission or chooses a degradation action. `finish=length` is the silent-truncation marker.

A.6 Position

For each trajectory we extract the answered batch indices. Because the model answers from the beginning of the list, the answered index distribution measures which parts of the input received processing.

A.7 Reproducibility

Tasks are generated by `experiments/batch_tasks.py` with fixed seeds. All calls use temperature 0. Raw JSONL and the scorer are in the repository.

A.8 Confidence intervals

With n=30 per condition, Wilson 95% intervals are roughly ±17 points at 50%. We emphasize directions and cross-model replication, not small differences.

References

G. R. J. Hockey. *Compensatory control in the regulation of human performance under stress and high workload: A cognitive-energetical framework*. Biological Psychology, 1997.

D. Kahneman. *Attention and Effort*. Prentice-Hall, 1973.

E. Hollnagel. *The ETTO Principle: Efficiency-Thoroughness Trade-Off*. Ashgate, 2009.

N. F. Liu, K. Lin, J. Hewitt, et al. *Lost in the middle: How language models use long contexts*. TACL, 2024.

A. Vaswani, N. Shazeer, N. Parmar, et al. *Attention is all you need*. NeurIPS, 2017.

Z. Ji, N. Lee, R. Frieske, et al. *Survey of hallucination in natural language generation*. ACM Computing Surveys, 2023.

E. M. Bender, T. Gebru, A. McMillan-Major, and S. Shmitchell. *On the dangers of stochastic parrots: Can language models be too big?*. FAccT, 2021.

S. Kadavath, T. Conerly, A. Askell, and others. *Language models (mostly) know what they know*. arXiv:2207.05221, 2022.

S. Farquhar, J. Kossen, L. Kuhn, and Y. Gal. *Detecting hallucinations in large language models using semantic entropy*. Nature, 2024.

G. Mialon, C. Fourrier, C. Swift, and others. *GAIA: A benchmark for general AI assistants*. ICLR, 2024.

S. Zhou, F. F. Xu, H. Zhu, and others. *WebArena: A realistic web environment for building autonomous agents*. ICLR, 2024.

C. E. Jimenez, J. Yang, A. Wettig, et al. *SWE-bench: Can language models resolve real-world GitHub issues?*. ICLR, 2024.

L. Kleinrock. *Queueing Systems, Volume 1: Theory*. Wiley, 1975.