

Yool Prompt vs Normal Instruction Benchmark

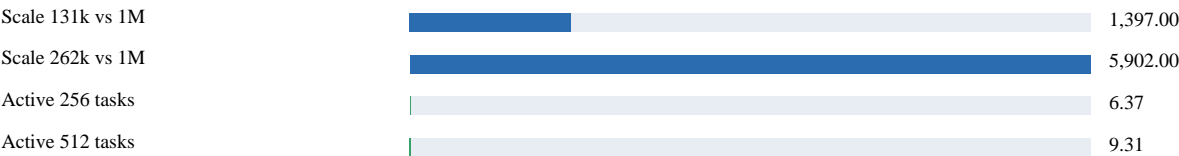
Repository: wesleysimplicio/simplicio-prompt

Run date: 2026-05-21

Executive Summary

The Yool prompt is faster because it turns a generic instruction into an execution protocol: lazy batch_spawn for scale, tuple-space routing, LaneWorkerPool fan-out, receipts, hookwall checks, and explicit runtime guardrails. The benchmark measured local runtime mechanics, not hosted LLM provider behavior.

Speedup (x)



Scale Representation

Profile	Represented agents	Wall time	Peak memory
Normal instruction, flat list	131,072	217.11 ms	28,749.88 KiB
Yool prompt, lazy batch_spawn	1,048,576	0.16 ms	6.32 KiB
Observed gain	8x more agents	1,397x faster	4,547x less memory

Profile	Represented agents	Wall time	Peak memory
Normal instruction, flat list	262,144	431.45 ms	57,542.32 KiB
Yool prompt, lazy batch_spawn	1,048,576	0.07 ms	6.39 KiB
Observed gain	4x more agents	5,902x faster	9,000x less memory

Peak memory reduction for scale representation (x)



Active Execution

Profile	Tasks	Wall time	Throughput	Peak memory
Normal sequential	256	603.98 ms	423.9 tasks/s	17.33 KiB
Yool lane fan-out	256	94.87 ms	2,698.3 tasks/s	879.82 KiB
Observed gain	same	6.37x faster	6.37x higher	higher active overhead

Profile	Tasks	Wall time	Throughput	Peak memory
Normal sequential	512	1,212.88 ms	422.1 tasks/s	33.55 KiB
Yool lane fan-out	512	130.28 ms	3,929.9 tasks/s	1,739.41 KiB

Observed gain	same	9.31x faster	9.31x higher	higher active overhead
---------------	------	--------------	--------------	------------------------

Token Usage and Estimated Savings

Token numbers are deterministic local estimates using `ceil(utf-8 bytes / 4)`. They are not provider billing measurements. The model compares repeated chat-context orchestration against one prompt plus compact tuple envelopes.

Scenario	Normal tokens	Yool tokens	Savings	Savings %
one_off_prompt_bootstrap	19	210	-191	-1005.26%
scale_1m_subagents	47,185,939	232	47,185,707	100.00%
active_256_tasks	11,520	6,610	4,910	42.62%
active_512_tasks	23,040	13,010	10,030	43.53%

Estimated token savings (%)



One-off bootstrap is intentionally more expensive with the Yool prompt, because it carries the execution protocol. Savings appear when work fans out: the protocol is paid once and subtasks travel as compact tuple envelopes.

Gains Beyond Speed

- Scalability: million-agent trees without flat lists.
- Auditability: tuple state plus receipts.
- Recovery: pending work can be replayed from tuple state.
- Failure isolation: lanes separate planning, build, test, review, and runtime work.
- Guardrails: CPU, queue, compression, hookwall, and disk GC are named controls.
- Portability: the same prompt points Claude, Codex, Hermes, and scripts at the same files.

Tradeoffs

For small active workloads, the Yool prompt uses more memory because it creates tuple envelopes and worker fan-out structures. It pays off when work is I/O-bound, uses subprocesses, browser automation, APIs, hosted LLM calls, or large-scale planning/execution.

Prompt Used

```
Use o repo canonico https://github.com/wesleysimplicio/simplicio-prompt.
Leia antes de editar: YOOL_TUPLE_HAMT.md, kernel/yool_tuple_kernel.py,
guardrails/cpu_throttle.py, guardrails/disk_gc.py, examples/python/receipts.py
e scripts/build_hamt.py.

Ao receber "Implement X": decomponha em grafo Hilbert-indexed, crie tuple raiz,
use batch_spawn(depth, branching, compression_threshold) para 1.000.000+
subagents sem enumerar, execute work ativo com spawn_agent, roteie por out/in/rd,
route_packet e scan_index, aplique hookwall, compress_token e prune_idle, e use
LaneWorkerPool respeitando YOOL_TUPLE_* env vars.

Execute:
python kernel/yool_tuple_kernel.py

Responda SEMPRE exatamente neste formato (sem variações):
[Tuple Space Snapshot]
[Active Agents/Subagents]
[Total Agents/Subagents]
[Próximo Yool a executar]
```

[Resultado parcial]

Reproduce

```
python benchmarks/prompt_vs_normal.py --json
python benchmarks/prompt_vs_normal.py --tasks 512 --scale-agents 262144 --sleep-ms 2 --json
python benchmarks/generate_prompt_benchmark_pdf.py
```