Nemotron 3.5 Super: *strong code and clean tool calls at full precision; long step-by-step state is where it slips.*

NVIDIA's Nemotron 3.5 Super — 120B parameters, 12B active per token, with a multi-token-prediction head built into the checkpoint — went through our full qualification pipeline at BF16 on four RTX PRO 6000 Blackwell cards: a sweep of its built-in speculative decoding, speed by context depth and by load, long-context retrieval, tool-call checks, and our code, math and agentic suites. Along the way we explain what each stage of the pipeline measures, and why it runs in that order.

Bottom line

- **It qualifies.** Every stage of our pipeline produced a verdict on four RTX PRO 6000s at full BF16 precision — after one re-run: the first concurrency curve was still climbing at its top load of 32 streams, so the pipeline refused to set a concurrency until we measured further.
- **Its built-in draft head is worth 3.2× on structured output and 1.8× on prose:** 411.2 and 230.6 tok/s for one user at draft depth 5, against 128.7 without speculation. Decode runs 365.1 tok/s at 1K of context and 157.8 at 120K; prefill holds 13.1K–14.5K tok/s from 4K to 120K.
- **1,476 tok/s across 64 parallel streams;** 56 streams get within 5% of that, with a new request's first token in 3.9 s.
- **Code and tool calls at full effort:** HumanEval 159/164, MBPP 478/500, byte-exact tool-call payloads 119/120; our hard agentic tier 25/34 and the repository tier 78/81, with none of its 764 tool calls unparsable. Tool-call checks clean on schema, tool choice, arguments and invented tools in all 44 runs.
- **The weak spot is long step-by-step state.** The ledger walk came out fully right 38/60 times (6 ran out of 32,768 tokens while still reasoning), and a 186-step decoyed chain through a long log was completed 3/18 times.
- **Medium effort is the better math setting:** 199/200 on our GSM8K slice against 196/200 at full effort, in 39 s instead of 222 s and with 3.3× fewer tokens. Thinking off: 191/200. (Measured on math only.)

1 · What we ran

- **Model:** `nvidia/NVIDIA-Nemotron-3.5-Super-VL-120B-A12B-BF16` @ `a1ccf94d` (OpenMDW-1.1 licence), 248.9 GB in 64 safetensors files. 120B parameters with 12B active per token: a hybrid of Mamba-2 and attention layers with mixture-of-experts, and a multi-token-prediction (MTP) head built into the checkpoint. It is a vision-language checkpoint; we ran it **text only** — image and video inputs set to zero, so the vision tower is never exercised.
- **Engine:** vLLM `0.32.0rc1.dev190+g3f1bcfd82` (torch 2.14, CUDA 13.0), the nightly build NVIDIA's model card names, pinned by image digest (`nightly-3f1bcfd82`, `sha256:cb9bc64f0f03…`); the release the card also cites

had not shipped. Flags from the card's four-GPU command: tensor parallel 4, async scheduling, Mamba shared-prefix checkpointing, the `qwen3_coder` tool parser and the `nemotron_v3` reasoning parser. Context 262,144 tokens, 64 sequence slots, GPU memory utilization 0.90.

- **Hardware:** our 4-GPU server — four RTX PRO 6000 Blackwell Workstation cards, 96 GB each — running our permanent overclock (GPC +350, MEM +6000). Every speed figure below includes it.
- **Setting:** thinking on at full effort, which is the chat template's default (`enable_thinking` true, `medium_effort` false), held fixed through every stage; greedy decoding (temperature 0); one run per cell. Medium effort and thinking off ran as controls on the math slice.
- **Tests:** the qualification pipeline (section 2), then HumanEval (164) and MBPP (500) executed with two rounds of self-repair, the *ledger walk* (60 long logs in which every running total must be carried), *payload fidelity* (120 tool calls whose file content must match a hostile text byte for byte), a seeded 200-item GSM8K slice, and two agentic tiers: 81 repository tasks and 34 held-out hard tasks, each scored by hidden tests.

---

2 · How the qualification pipeline works

Qualifying a model is one fixed sequence of stages, ordered so that cheap decisions come before expensive measurements: first choose the serving config, then characterise that config, then measure what it can do. One command drives it. It works out where it is from the files each stage leaves behind and stops at every point that needs a human decision — downloading, and launching servers. Each artifact is stamped with the procedure that produced it, and a final checker refuses the set if a stage is missing.

Stage	What it measures	Why it is there
Fit check	The repository's metadata, before any download: file types, unsafe formats, the exact revision. Here 64 safetensors files, 0 unsafe files, revision `a1ccf94d`.	Refuses a model before hours of download, and pins the exact weights every later number describes.
Serving profile	A reviewed launch template, pinned to an image digest, that declares which serving options are worth sweeping — here the depth of the built-in MTP draft head: 1, 2, 3, 5.	What a model supports cannot be read reliably from its metadata, so it is declared once and reviewed; every launch is rendered from that template, never typed by hand.
Speculation sweep	A fresh server for each declared option plus a no-speculation baseline; one-user decode on a JSON prompt and a prose prompt, 5 repeats each; the KV-cache size the engine reports for each.	How many draft tokens are accepted depends on how predictable the text is, so one prompt would tell whichever story it happened to tell. The serving config is chosen before anything expensive is measured.
Lock	A record of the winning option and its measured tokens per decoding step (5.19 on JSON). The depth ladder and the concurrency stage first measure the running server and refuse a mismatch; here it read 5.00, inside tolerance.	Choosing a config is not the same as serving it; the fingerprint catches a server left on the wrong one.
Depth ladder	22 prompt lengths from 1K to 120K tokens, 5 repeats: prefill from the first, cache-cold repeat, decode as the median. Then a retrieval check at the deepest length: two linked facts at 5% and 92% depth.	Speed at 1K says little about speed at 100K. Every length gets a fresh, salted prompt so a cached prefix cannot fake a fast prefill.
Concurrency	1 to 64 simultaneous agent-shaped requests (11,222-token prompts, 250 tokens out), 3 repeats; total and per-stream throughput, and the time to first token of one extra request fired from a separate process while the load runs.	Sets how many parallel workers to run. The knee is the lowest load within 5% of the best total; a knee on the top tested load is censored and gets no recommendation.
Needle grid	Retrieval at 6 lengths from 8K to 120K tokens across 6 variants, from one planted fact to a 186-step decoyed chain, 3 repeats. A reply cut off by the output limit is recorded as its own state, never as a wrong answer.	Finding one fact at depth is easy; following, counting and refusing look-alikes at depth is not.
Tool checks	22 tool-use cases in 8 categories, 2 repeats: schema validity, tool choice, argument values, invented tools, recovery from an error, asking instead of inventing a missing value.	A fabricated argument arrives perfectly well-formed; only the call log shows it was never supplied.
Quality and agents	HumanEval and MBPP (executed, with two rounds of self-repair), the ledger walk, payload fidelity, a GSM8K slice under three thinking settings, and two agentic tiers in sealed containers with no network.	Run against the same locked server at the same thinking setting, so the capability and the speed describe one config.
Coverage check	Reads the provenance stamp on every artifact and fails the run set if any stage is missing, unstamped or ran without a verdict.	It failed this run once (1 stage without a verdict: the censored concurrency curve) and passed after the re-run.

The point of the order is that every later number describes the config that will actually be served. Two of the guards are visible in this run: the lock check, which passed (5.00 tokens per step measured against 5.19 locked), and the concurrency stage, which refused to recommend anything from its first curve (section 3).

3 · Speed: the draft head, context depth and load

The speculation sweep

Server config	JSON, tok/s	Tokens per step	Prose, tok/s	Tokens per step	KV cache, tokens
No speculation	128.7	1.03	128.7	1.02	6.47M
MTP, depth 1	225.9	1.95	212.6	1.83	5.24M
MTP, depth 2	243.4	2.89	197.3	2.33	5.08M
MTP, depth 3	334.6	3.66	226.1	2.46	4.97M
MTP, depth 5 (locked)	**411.2**	5.19	**230.6**	2.89	4.76M

One stream, 384 tokens, median of 5; thinking on, greedy. Tokens per step = tokens emitted per decoding step: about 1 without speculation, at most one more than the draft depth with it. KV cache as the engine reports it at start-up. vLLM 0.32.0rc1.dev190+g3f1bcfd82 (nightly, pinned by image digest), BF16 weights, tensor parallel over four RTX PRO 6000 Blackwell cards running our permanent overclock (GPC +350, MEM +6000).

- **Depth 5, the model card's own value, was fastest on both prompts and was locked:** 3.2× on JSON and 1.8× on prose against no speculation. JSON speed was still rising at depth 5; we did not try deeper drafts.
- **Structured text is far easier to predict than prose:** 5.19 against 2.89 tokens per step at the same depth — which is why the sweep runs both, and why the speed you see depends on what you ask for.
- **The draft head costs KV cache:** 6.47M tokens without it, 4.76M at depth 5 — still room for 18 requests at the full 262,144-token context.

By context depth

Prompt tokens	Prefill, tok/s	First token	Decode, tok/s	Tokens per step
1,105	10,172	0.11 s	**365.1**	5.33
2,064	11,744	0.18 s	365.3	5.45
4,044	13,613	0.30 s	346.4	5.22
8,048	14,543	0.55 s	344.7	5.33
16,090	13,827	1.16 s	331.7	5.45
32,112	13,884	2.31 s	278.5	5.12
64,156	13,673	4.69 s	207.6	4.66
100,077	13,368	7.49 s	172.2	4.66
120,126	13,148	9.14 s	**157.8**	4.74

Prefill = prompt tokens ÷ time to first token on the first, cache-cold repeat; decode = median of 5 repeats of 256 tokens (repeats 2–5 reuse the cached prompt by design). 22 lengths were measured; 9 are shown. Thinking on, greedy. vLLM 0.32.0rc1.dev190+g3f1bcfd82 (nightly, pinned by image digest), BF16 weights, tensor parallel over four RTX PRO 6000 Blackwell cards running our permanent overclock (GPC +350, MEM +6000), MTP draft depth 5.

- **Prefill is flat:** 13,148–14,543 tok/s from 4K to 120K tokens; a 120,126-token prompt reaches its first token in 9.1 s from cold.
- **Decode falls with depth**, from 365.1 tok/s at 1K to 157.8 at 120K — 43% of its short-context speed. On the ladder's prompt the draft head accepted 4.27–5.57 tokens per step — near the sweep's JSON figure and well above its prose figure (2.89) — so expect free-form prose to run slower than this column at the same depth (at short context it ran 230.6 tok/s).
- **The retrieval check at 120,246 tokens passed:** both planted facts came back, linked correctly.

Many users at once

Parallel streams	Total, tok/s	Per stream, tok/s	New request's first token
1	280	279.7	0.04 s
2	408	220.2	0.04 s
4	590	183.7	0.04 s
6	685	149.1	0.10 s
8	772	128.6	0.12 s
12	952	103.2	0.10 s
16	1,034	91.5	0.13 s
24	1,166	69.3	0.48 s
32	1,264	56.2	1.25 s
40	1,325	50.6	2.13 s
48	1,383	43.3	2.96 s
56 (knee)	1,422	38.7	3.85 s
64	1,476	35.0	8.08 s

Agent-shaped requests: 11,222-token prompts sharing a cached prefix, 250 tokens out, thinking on, greedy; median of 3 repeats (the first repeat at each load was always the slowest, by up to 36%). The last column is one extra short request sent from a separate process during the load. Server: 64 sequence slots. vLLM 0.32.0rc1.dev190+g3f1bcfd82 (nightly, pinned by image digest), BF16 weights, tensor parallel over four RTX PRO 6000 Blackwell cards running our permanent overclock (GPC +350, MEM +6000), MTP draft depth 5.

- **Total throughput climbs all the way to 1,476 tok/s at 64 streams**, the server's slot limit. The knee — the lowest load within 5% of the best — is 56: each stream still gets 38.7 tok/s and a new request waits 3.9 s for its first token. At 64 it waits 8.1 s.
- **The first run stopped at 32 streams** and its best total (1,260 tok/s) was on its top load. The pipeline reported the curve as censored and refused a recommendation, and the coverage check failed the run set until we re-ran the stage up to 64.

- **Our rule** — the knee, capped at the slot count minus two kept free for interactive requests — gives 56 parallel workers on a 64-slot server.

4 · Quality at full effort, and what medium effort changes

Test	Passed	Cut off	Score on finished	Median tokens
HumanEval	159 of 164	0	97.0%	898
MBPP (tasks 11–510)	478 of 500	10	97.6%	558
Ledger walk, fully right	38 of 60	6	83.2% [1]	11,932
Payload, byte-exact	119 of 120	0	99.2%	482
GSM8K slice	196 of 200	1	98.5%	350

Thinking on at full effort, greedy, 32,768 tokens of room per answer, one run. Passed counts every item, so a cut-off reply counts as failed; the score leaves cut-off replies out. Median tokens = generated tokens per item, reasoning included. [1] For the ledger walk the score is partial credit: the mean share of each walk written correctly before the first wrong line.

- **Self-repair:** shown its failing test, it fixed 4 of 5 HumanEval failures and 9 of 12 MBPP failures within two rounds.
- **All 17 cut-off replies spent the whole 32,768-token budget reasoning and wrote no answer at all** — reasoning that did not converge, not long answers.
- **The one payload miss** first diverged from the original at character 24 of 364.

Where the ledger walk falls off

Log	Steps	Operations	Fully right	Cut off
8K	12	add	8 of 10	1
8K	20	add, subtract, XOR	8 of 10	0
16K	20	add	10 of 10	0
16K	30	add, subtract, XOR	6 of 10	2
32K	30	add	5 of 10	0
32K	40	add, subtract, XOR	1 of 10	3

Each step of the walk is one node in a long log; the model must write every node and its running total, in order, so one slip spoils every later line. Prompts 9,230–35,764 tokens.

Accuracy falls as the log grows and the operations mix: from 10 of 10 on 20 additive steps through a 16K log to 1 of 10 on 40 mixed steps through a 32K log. Of the 16 finished walks that went wrong, 11 lost the path (stopped finding the next node) and 5 kept the path but let a running total drift — all 5 in the mixed-operation sets.

Thinking effort on math

Setting	Correct	Cut off	Median tokens	Reasoning, p95 characters	Tokens, whole slice	Wall time
Full effort (template default)	196 of 200	1	350	9,899	216,118	222 s
Medium effort (`medium_effort`)	**199 of 200**	0	231	1,617	64,719	39 s
Thinking off	191 of 200	0	89	0	18,922	9 s

Seeded 200-item GSM8K slice, greedy, 32 requests in parallel against the same server. Output room 32,768 tokens with thinking, 16,384 without. The medium switch adds 7 tokens to every prompt and thinking off left 0 of 200 replies with any reasoning, so both settings reach the template. vLLM 0.32.0rc1.dev190+g3f1bcfd82 (nightly, pinned by image digest), BF16 weights, tensor parallel over four RTX PRO 6000 Blackwell cards running our permanent overclock (GPC +350, MEM +6000), MTP draft depth 5.

Medium effort scored as well as full effort — 199 against 196, a gap inside run-to-run noise — and finished the slice 5.7× faster; its 95th-percentile reasoning is 6.1× shorter. Thinking off costs 5 problems of 200. We measured the effort switch on math only; code, the ledger walk, payloads and both agentic tiers ran at full effort.

5 · Agentic work and tool calls

Both agentic tiers run the model as a coding agent in a sealed container with no network. It gets a small repository and a task and works through tools — list, read, write and edit files, run Python and run the visible tests — for up to 25 steps; a hidden test suite decides pass or fail. The hard tier is built so the visible tests are not the whole specification.

Hard tier (34 held-out tasks), by family	Passed
Trap, open-book: the rule is written down in the repository	**10/10**
Spec completion: implement a small format or parser from its written spec, against hidden test suites	5/8
Performance: make a slow routine fast without changing a single output	**2/2**
Wrong-file symptom: the error shows in one file, the cause sits in another	**4/4**
Trap, closed-book: no spec; the model must know the unstated standard	4/10
Hard tier (tier 3), total	**25/34**
Repository tier (tier 2): bug fixes, features, reports, messy data	78/81

Thinking on at full effort, JSON tool calls, 32,768 tokens per turn, greedy, one run.

- **No loops, no cut-offs:** a median of 6 steps per hard task (most 10) and 7 per repository task (most 13), of 25 allowed. Every task ended with the model stopping on its own; the misses are wrong solutions, not runs that hit a limit.
- **Where it misses:** 6 of its 9 hard-tier misses are closed-book traps — the rule is not in the repository and the model has to know the convention; the rest are spec-completion tasks. Its 3 repository-tier misses are all messy-data tasks.

Tool check	Runs	Clean on all four checks	Behaviour
Simple	10	**10**	one correct call, 10 of 10
Sequential	4	**4**	used the first result for the next call, 4 of 4
Parallel	4	**4**	made every lookup asked for, 4 of 4
Missing parameter	6	**6**	made no call and asked for the value, 6 of 6
No such tool	6	**6**	said the capability does not exist, 6 of 6 (2 after first checking the VM and the host)
Error recovery	6	**6**	corrected the name and retried, 4 of 6; asked the user to confirm the suggested name, 2 of 6
Ambiguous	4	**4**	committed to one acceptable tool, 4 of 4
Nested	4	**4**	used the first result for the next call, 4 of 4

2 repeats of 22 cases, 4,096 tokens per turn, thinking on, greedy. The four checks: arguments match the schema, the right tool, the right argument values, no tool name outside the provided set. In one nested case (2 runs) our scorer looked for the digits without a thousands separator and the model wrote them with one; read by hand, the answer is right.

Asked for something with a value missing, it asks instead of inventing one. Inventing is the failure that matters most in an agent loop, because a made-up argument reaches the target perfectly well-formed. On a mistyped VM name it either retried with the suggested name or asked to confirm it; asking is the safe choice, but in an unattended loop it stalls.

6 · Long-context retrieval

Variant	8K	16K	32K	64K	100K	120K
One fact, at 5 depths	**5/5**	**5/5**	**5/5**	4/5 · 1 cut	**5/5**	**5/5**
One fact among look-alikes	**3/3**	**3/3**	2/3 · 1 cut	**3/3**	**3/3**	2/3 · 1 cut
Three linked facts	**3/3**	**3/3**	**3/3**	**3/3**	**3/3**	**3/3**
Sum of 5 scattered values	**3/3**	**3/3**	**3/3**	**3/3**	**3/3**	**3/3**
15 of 200 planted facts	**3/3**	**3/3**	**3/3**	**3/3**	**3/3**	**3/3**
186-step decoyed chain	**0/3**	**0/3**	**0/3 · 1 cut**	**1/3 · 2 cut**	**1/3 · 1 cut**	**1/3**

Found / tries; "cut" = replies stopped by the output limit (24,000 tokens, 36,000 for the chain). Length = the log before the planted lines are added: prompts ran 8,118–142,618 tokens, the chain's 30,505–142,618. 3 tries per cell; the single-fact row is one try at each depth. Thinking on, greedy.

- **Outside the chain, every miss was a reply that ran out of room before writing any answer** (3 of 102); none answered wrong, and the look-alike variant never returned a look-alike (0 decoy hits in 18 tries).
- **The chain is the hard one:** 186 steps, each with 3 near-miss decoys, scattered out of order through the log. Complete walks: 3 of 18 — 0 of 9 at 8K–32K, 3 of 9 at 64K–120K. 4 replies ran out of room before writing any of

the walk; the other 11 stopped on their own after 11.1K–33.6K tokens with the walk wrong partway (1–157 of 186 steps right). With 3 tries per cell, the better showing at long lengths is not a reading.

7 · What to run

- **Serving:** vLLM `0.32.0rc1.dev190+g3f1bcfd82`, pinned by digest; BF16 with tensor parallel 4 across four 96 GB cards (the weights are 248.9 GB), the card's flags, and `--speculative-config '{"method":"mtp","num_speculative_tokens":5}'` — 1.8× on prose, 3.2× on structured output.
- **Context and slots:** 262,144 tokens and 64 slots fit with 4.76M tokens of KV cache at memory utilization 0.90.
- **Agent fleets:** up to 56 parallel workers on 64 slots. Past the knee, total throughput still rises a little, but a new request waits 8.1 s for its first token at 64.
- **Thinking:** for math-style work, send `"medium_effort": true` in the chat-template arguments — as accurate here, at 3.3× fewer tokens. For code and agents we measured full effort only, and it did well.
- **Long running state:** budget for long reasoning — the ledger walk's median reply was 11,932 tokens and 6 of 60 ran past 32,768 without answering — and check long running totals it hands back.
- **Unattended loops:** it asks rather than guessing when a value is missing or a name is wrong; make sure your loop can answer.

8 · Caveats

- One run per cell, greedy decoding, not the model card's recommended sampling. On a batched server even greedy runs vary from run to run, so gaps of a few items are noise.
- Thinking was held at the template default (full effort) for every stage; medium effort and thinking off were measured on math only.
- Text only: the checkpoint's image and video inputs were disabled and not measured.
- Served at 262,144 tokens of context; the depth ladder and retrieval grid stop at about 120K tokens. Longer contexts were not measured.
- Draft depths 1, 2, 3, 5 were tried; JSON speed was still rising at 5. Concurrency throughput was still rising at 64 streams, the slot limit we served; a larger slot count was not measured.
- Our agentic runner sends each earlier turn back without its reasoning, as most agent frameworks do. For one model we tested this decided the result (Recursive Flint); we have not run Nemotron 3.5 Super with its reasoning sent back.
- The retrieval check at 120K answered correctly, then repeated the same answer line: the recorded end of its reply holds 4 identical copies.
- The decoyed chain had 3 tries per cell; read its depth pattern as anecdote.
- The four RTX PRO 6000s run a permanent overclock (GPC +350, MEM +6000); absolute speeds include it.
- The engine is a nightly build; numbers may move on the release NVIDIA's card cites.
- Not measured: NVIDIA's quantized (NVFP4) variants, other engines, images and video, languages other than English, power draw.

Every figure on this page was computed from the raw rows of runs made on 15 October 2026 on our own machines. Run directory: `bench/runs/20261015-nemotron35-super` (the sweep and its lock, the depth ladder, both concurrency runs, the retrieval grid, the

tool checks, and the item-level rows of the quality, math and agentic sets). The rows are kept; to see them, or to send a model you want measured, write to hello@netsphere.com.ai. Online: netsphere.com.ai/reports/nemotron-3-5-super/