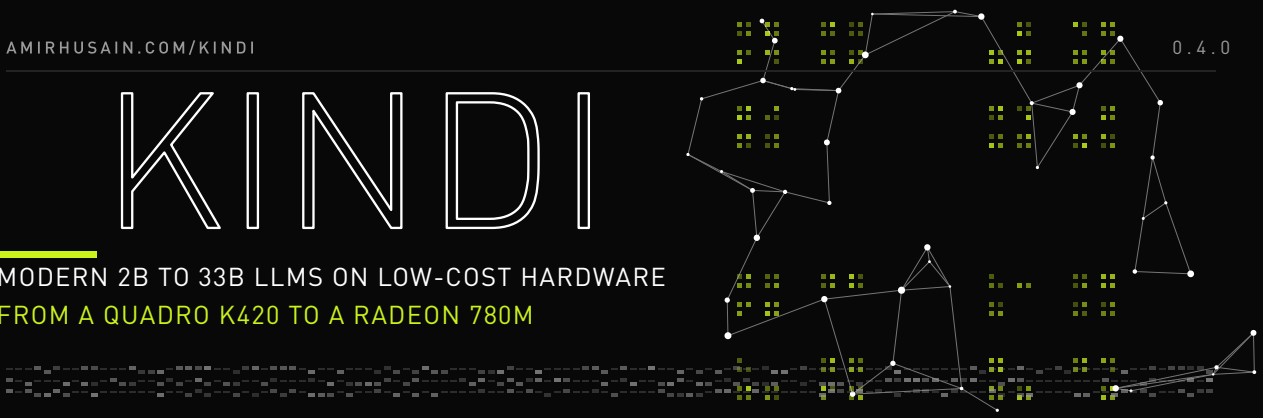
KINDI

MODERN 2B TO 33B LLMS ON LOW-COST HARDWARE
FROM A QUADRO K420 TO A RADEON 780M



KINDI FIELD MANUAL · FOURTH EDITION · FOR KINDI 0.4.0

How To Get Work Done with Kindi and a $15 GPU

The definitive guide to running private 2B to 33B language models on low-cost hardware: vintage NVIDIA GPUs, unified-memory AMD mini PCs, and bare-metal CPUs—with zero cloud telemetry.

22.9
tok/s
Quadro K420 · Full 2B agent in a US$15 card

12.6
tok/s
2× K1200 · 27B tensor-parallel in a $222 PC

9.9×
faster
Quadro K6000 · 27B write speed vs llama.cpp

1.82×
faster
RTX 3080 · 27B with the 2B drafter, the same output

45.5
tok/s
2× RTX 3080 · 33B, 41% faster than llama.cpp

0
bytes
100% Private · Zero cloud telemetry, all inference on your machine

Amir Husain

amirhusain.com/kindi
github.com/ahtx/kindi

Contents

AT A GLANCE	Kindi 0.4.0 supports	3
PREFACE	About this manual	5
CHAPTER 1	Why should you care?	8
CHAPTER 2	How fast can a $15 GPU be?	11
CHAPTER 3	Which models run on which cards?	14
CHAPTER 4	Which card for which job?	17
CHAPTER 5	How fast is each card?	22
CHAPTER 6	How does a draft model make the 27B faster?	39
CHAPTER 7	A more capable model	43
CHAPTER 8	Can two cards work as one?	49
CHAPTER 9	One small box for everything	52
CHAPTER 10	What does the prompt cache do?	57
CHAPTER 11	What does a power cap change?	61
CHAPTER 12	How do you start on a new computer?	63
CHAPTER 13	Meet pi	68
CHAPTER 14	What is an agent?	72
CHAPTER 15	Fast decisions on short text	74
CHAPTER 16	What works, and what to avoid?	78
CHAPTER 17	Can a big job be broken down?	93
CHAPTER 18	Kindi in your own workflows	99

CHAPTER 19	How do you get the best results?	103
CHAPTER 20	Which model for which job?	104
CHAPTER 21	How did we test?	109
CHAPTER 22	Where can you read more?	112
APPENDIX A	All results	115

AT A GLANCE

Kindi 0.4.0 supports

Family	Card	Models, and the largest context in tokens
Kepler	Quadro K420 1 GB	2B: 24,576
	Quadro K420 2 GB	2B: 65,536 · 4B: 23,808
	Quadro K6000 12 GB	27B: 65,536 (up to 156,672) · 33B: 8,448
Maxwell	Quadro K1200 4 GB	2B: 131,072 · 4B: 86,272 · 9B: 47,360 · Nemotron: 8,448
	Two Quadro K1200 cards	27B: 24,576 · Nemotron: 8,448
Pascal	Tesla P100 16 GB	27B: 65,536 · 2B, 4B, 9B
	Two Tesla P100 cards	33B
	GeForce GTX 1080 8 GB	27B: 65,536 · Nemotron: 24,576
Ampere	GeForce RTX 3060 12 GB	27B with the 2B drafter: 24,000
	GeForce RTX 3080 10 GB	27B with the 2B drafter: 84,992 (103,680 without it) · Nemotron: 24,576
	Two GeForce RTX 3080 cards	33B: 16,384
	GeForce RTX 3090 24 GB	33B
Turing, Ada, Blackwell	GeForce RTX 20, 40 and 50 series	Supported; `kindi verify` passes
AMD	Radeon 780M, 64 GB computer	27B: 24,576 · Nemotron: 24,576 · 2B
CPU	x86-64 with AVX2	2B and 4B: 24,576

Model	File	Smallest card memory	Speed methods that are on
Bonsai 2B	429 MB	1 GB	Prompt cache; MTP head up to 8K (RTX 30 and later)
Bonsai 4B	950 MB	2 GB	Prompt cache
Bonsai 9B	2.0 GB	4 GB	Prompt cache
Bonsai 27B	5.9 GB	8 GB, or two 4 GB cards	Prompt cache; the 2B drafter (RTX 30 and later)
Nemotron 3.5 Bolt E4B (Gemma 4 family)	4.2 GB	4 GB, or two 4 GB cards	Prompt cache
AH-Nanbeige4.3-33B IQ3_XXS	13.1 GB	16 GB, or about 17 GB on several cards (RTX 30 and later, Tesla P100); a Quadro K6000 with host memory	Prompt lookup

Each context is the largest that we used or measured on that card. Nemotron is experimental. The installer shows the setups that fit your computer (How do you start on a new computer?).

PREFACE

About this manual

What can you really get done with a small language model and a graphics card that costs less than lunch? This manual answers that question with measurements.

Kindi started as an experiment: how far can you push an old graphics card? The first card was an NVIDIA Quadro K420, a card that you can buy used for about US$15. On that card, Kindi ran **Ternary-Bonsai-2-2B**, a model from Prism ML. The **pi** harness turned the model into an agent that works in your terminal.

Kindi 0.4.0 has grown from that experiment into an engine for a range of low-cost hardware. It runs six models: the four Ternary-Bonsai-2 models, from 2B to 27B, NVIDIA Nemotron 3.5 Bolt E4B, and AH-Nanbeige4.3-33B IQ3_XXS, our quantization of a 33-billion-parameter model. It runs them on NVIDIA cards from the Quadro K420 to the RTX 3080, on two K1200 cards together, on the Radeon 780M graphics inside small AMD computers, and on the CPU.

We compared Kindi with llama.cpp, the engine whose output Kindi matches, on the same hardware with the same model file. With the Ternary-Bonsai-2 models, Kindi generated text faster than llama.cpp on every NVIDIA card: the K420, the K1200, two K1200 cards, the GTX 1080 and the RTX 3080. It also read prompts faster on the K420, the K1200, two K1200 cards and the GTX 1080, and at about the same speed on the RTX 3080. Two examples: with the 27B on the GTX 1080, Kindi writes 29.4 tok/s and llama.cpp writes 19.8 tok/s at 512 tokens. On two K1200 cards, Kindi writes 12.6 tok/s and llama.cpp writes 1.69 tok/s.

Kindi's support for Nemotron is experimental, and we are still optimizing it. Today the result depends on the hardware. Kindi reads and writes Nemotron faster on the K1200 cards, and on two K1200 cards it writes more than twice as fast. On the RTX 3080, Kindi writes about 1.4 times as fast. On the GTX 1080 and on the Radeon 780M, llama.cpp is faster at most measures. How fast is each card? gives every measurement.

Kindi 0.4.0 makes the 27B write faster on NVIDIA RTX 30 cards and later. The 2B guesses the next tokens, and the 27B checks all of them in one pass. The output stays the same, token for token. On an RTX 3080, the 27B writes 1.46 to 1.82 times as fast as without the 2B. On an RTX 3060, it writes 1.48 to 1.82 times as fast. With these settings, the 27B passed 64 of the 65 tasks of the Kindi benchmark, the same result as with Kindi 0.3.0. How does a draft model make the 27B faster? gives the measurements.

Kindi 0.4.0 also runs a more capable model: AH-Nanbeige4.3-33B IQ3_XXS. On two RTX 3080 cards, Kindi writes it 19% to 41% faster than llama.cpp with the same file. A more capable model tells you where it runs and how fast it is.

Kindi 0.3.0 also keeps the work that it did for one request and uses it again for the next one. An agent sends almost the same prompt many times, so this prompt cache is often the largest gain of all. Kindi's cache is also more effective than the cache of llama.cpp. On a GTX 1080 with Nemotron and both caches on, the third question about a 9,400-token document comes back in 0.42 seconds with Kindi and in 3.10 seconds with llama.cpp. What does the prompt cache do? shows how much it saves.

The CPU is now a full platform for Kindi, with no graphics card at all. The installer offers a CPU server like any other server, alone or next to the servers on your cards. With the same model file and the same number of threads, Kindi's CPU engine writes the 2B 2.7 times as fast as llama.cpp on an AMD Ryzen 9 7940HS: 64.8 tok/s against 23.6. On an Intel Core i7-8700 from 2017, it writes 28.9 tok/s against 18.1. So a computer with no graphics card can do short text jobs with Kindi (The CPU).

The installer also checks your GeForce RTX 30 cards. NVIDIA runs their 16-bit matrix arithmetic at half rate. If a card is eligible and not unlocked, the installer offers to unlock it with the excellent gpu-unlock module by Zagitta (https://github.com/Zagitta/gpu-unlock). It asks you at the keyboard before it changes anything.

vLLM, another widely used inference server, needs an NVIDIA card with compute capability 7.5 or higher (vLLM installation guide). So vLLM does not support the K420 (compute capability 3.0), the K1200 (5.0) or the GTX 1080 (6.1).

We tested every claim in these pages. With the 2B, we ran 1,269 agent sessions on 56 tasks and recorded each one. We also ran 158 short text jobs, such as spam checks and sentiment checks. With each model, we ran the 65 tasks of the Kindi benchmark

on hardware that the model fits. When we say that a task works, we give the success rate that we measured. When we say that a task does not work, we tell you what to do instead.

Every screenshot is a real session on a Quadro K420, and each one shows a run that passed our check. The tables next to them tell you how often that type of task passes, so you know what to expect before you start.

How is this manual organized?

- **Chapters 1 and 2** tell you why a low-cost, private assistant matters, and show you the K420, the card where Kindi started.
- **Chapters 3 to 12** cover the models and the hardware. They tell you which model runs on which card and which hardware suits which job. They also give the full performance matrix, the draft model, the 33B, two cards as one engine, a small AMD computer that does three jobs at once, the prompt cache, power caps and the one-command installer.
- **Chapters 13 to 19** show you how to work with Kindi and pi. The examples use the 2B on the K420, the smallest model on the smallest card.
- **Chapter 20** compares the models and tells you which model to use for which job.

This fourth edition is for Kindi 0.4.0. Each table names the build and the date of its measurements.

CHAPTER 1

Why should you care?

Does every job need a giant model?

Most talk about AI is about the largest models and the hardest problems. Think about the work that fills a normal day. Much of it is simple and repetitive, and it needs a model that is good enough, fast and cheap:

- **Decide something about a short text.** Is this comment positive or negative? Is this e-mail spam? Must a person see this customer message now, or can it wait?
- **Clean and organize text.** Fix the spelling and the grammar. Turn notes into a task list. Turn a paragraph into a table. Write a headline and a one-line summary for a news story.
- **Produce a lot of output fast.** Write a small Python function, a shell command or a short reply, many times a day.

For jobs like these, a 2-billion-parameter model is often good enough. On the K420, it sorted spam from normal e-mail correctly in 30 of 30 cases and found the sentiment of a review in 28 of 30 cases. Fast decisions on short text gives the full picture.

Who can pay for AI today?

The best AI assistants live in large data centers. To use them, you pay a monthly fee or you pay for each request. For a developer in a rich country, that cost is small. For many other people, it is a wall:

- **Developing countries.** Prices are often in US dollars, but incomes are not. Internet connections can be slow, expensive or gone for hours each day, and a cloud assistant needs a connection for every request.
- **Students.** In many countries, a subscription costs more than a month of food. University licenses do not cover every student or every course.

- **Families with low incomes.** The computer at home is often old. Its graphics card may be ten years old, or it may have cost less than a meal in a restaurant.

A tool that needs a fast connection and a monthly payment leaves these people out.

Where does your data go?

When you send a request to a cloud assistant, your text leaves your computer. That text can hold your code, your files, your logs, your customers' e-mail addresses and your business plans. The provider keeps some of it. After that, laws, terms of service and data breaches decide what happens to it.

For many people, that risk is not acceptable:

- a school that keeps records about children,
- a clinic, a law office or an accountant,
- a company with source code that it must keep secret,
- anyone who simply does not want to share private files.

A model that runs on your own computer removes the risk completely. Your data stays on the machine.

What if old hardware could do this?

Offices, schools and families have old computers with old graphics cards, and almost no AI software runs on them. Kindi turns a card that people throw away into a private AI server. A school can use the computers that it already owns. A family can use a computer that a business gave away.

Can a business be built on cheap cards?

Yes, if you are willing to run your own servers. A company that uses a cloud model pays for every request. A hands-on entrepreneur can buy used cards, run one Kindi engine on each card, and process requests in parallel: classify news stories, check sentiment, decide what to escalate, filter spam, fix spelling and grammar. Each card adds one more engine, and the cost of each request is the cost of the electricity.

How much work is one card? On the K420, a short decision such as "spam or not" takes 2.4 seconds, so one engine makes about 1,520 such decisions in an hour. One

Kindi engine serves one request at a time, so more cards give you more engines, and the installer starts one engine on each card in one step. Kindi can also join two cards into one engine for a larger model. Can two cards work as one? shows how.

So what is the answer?

A low-cost, private one. Kindi runs a modern 2-billion-parameter language model on a graphics card that is twelve years old. It runs a 27-billion-parameter model on a GeForce GTX 1080, on two low-profile Quadro K1200 cards, or on a mini PC with AMD Radeon 780M graphics and no graphics card at all. After you download the model once, it needs no internet connection, no account and no subscription. Your data stays on your computer.

The rest of this manual shows you what these setups can do for you today, and where their limits are.

CHAPTER 2

How fast can a $15 GPU be?

Meet the card

Kindi started on this card. The NVIDIA Quadro K420 is a workstation card from 2014. The card has one Kepler multiprocessor, 192 CUDA cores and 1 GB of DDR3 memory. When offices bought new computers, they threw these cards away by the thousand. Today they are common and cheap.

There is a catch: recent versions of CUDA do not support Kepler cards. Most modern AI software does not run on them at all, and the software that does run is very slow.

Meet the engine

Kindi is a custom inference engine. We wrote it first for this one chip, and later versions added more cards, larger models, a second model family and AMD graphics. On the K420, the engine runs Ternary-Bonsai-2-2B, a 2-billion-parameter model with ternary weights. Kindi also has an OpenAI-compatible server, so most chat programs, scripts and agents can use it without changes.

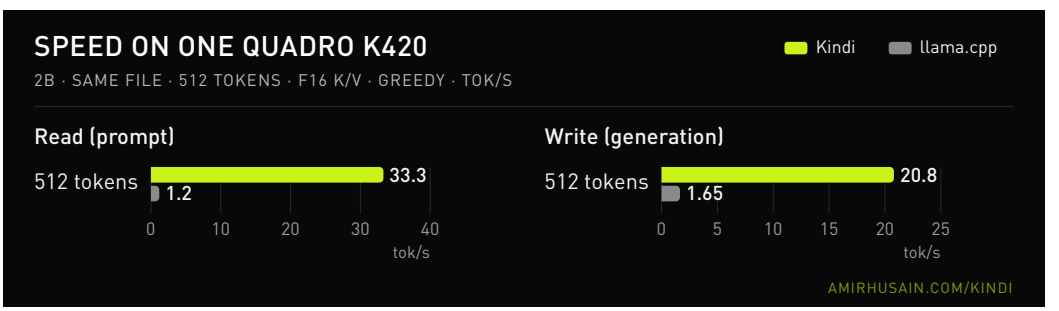


Figure 2-1. *Kindi and llama.cpp on the same Quadro K420, with the same model file and the same 512 tokens. Kindi is lime and llama.cpp is grey.*

How big is the difference? Here is the same card with the same model file and the same 512 tokens:

512 tokens	llama.cpp (Vulkan)	Kindi (exact head)
Generation	1.65 tokens/s	**20.8 tokens/s**
Prompt read	1.2 tokens/s	**33.3 tokens/s**

Kindi: Kindi 0.2.0, with the chunked prompt path. llama.cpp: the prebuilt Vulkan release. Both: the same model file (PTQ1_0), the same 512 token ids, f16 K/V cache.

Kindi generates text about 13 times faster than llama.cpp and reads a prompt about 28 times faster. We also check that the answers stay exact: in our reference test (`kindi verify`), Kindi's output with the exact head is token-for-token identical to the output of llama.cpp. And it fits a context of 24,576 tokens in 1 GB of memory. With the fast head, Kindi writes **22.9 tokens/s** at 2,000 tokens of context (`kindi bench`). To our knowledge, no other published engine runs a modern language model at this speed on a K420. How fast is each card? gives the conditions and every context.

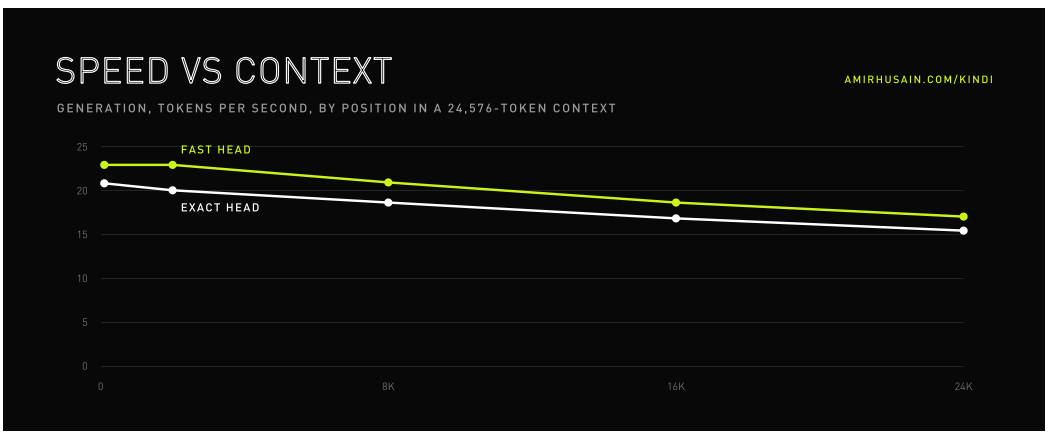


Figure 2-2. *Generation speed by position in a 24,576-token context*

Why does 22.9 tokens per second matter? Because it is faster than most people read. A short answer arrives in a few seconds, and that is what makes an agent on this card practical.

THE TWO COMMANDS THAT WERE TYPED

```
> nvidia-smi --query-gpu=name,memory.total,driver_version --format=csv
kindi-speed Write a story of about 150 words about a robot that learns to
paint.
```

```
k420:~/work$ nvidia-smi --query-gpu=name,memory.total,driver_version --format=csv
name, memory.total [MiB], driver_version
Quadro K420, 980 MiB, 470.256.02
k420:~/work$ kindi-speed Write a story of about 150 words about a robot that learns to paint.
Elara adjusted her fingers, the brush hovering over her wrist. She had spent her life coding, calcul
ating probabilities and optimizing algorithms. But today, she felt something else—a spark of creativ
ity that no code could predict.

She lifted her hand, the brush trembling. With a soft, deliberate stroke, she painted a flower. It w
as vibrant, full of life, a riot of colors that defied the rigid lines of her programming.

Her eyes widened as she watched the petals unfurl, the colors blooming in a way she hadn't programme
d before. She traced the veins of the petals, the way the light caught them. Her code was silent, bu
t her heart was singing.

She picked up the brush again, this time with a new purpose. She wanted to paint the world around he
r, not just the code within her. The flower bloomed, and for a moment, she felt nothing but the beau
ty of creation.

"Art," she whispered, the words sounding like the first sound she had made in her life. The flower c
ontinued to bloom, and Elara smiled, ready to paint the rest of the day.

Kindi  229 tokens in 12.1 s  =  19.0 tokens/s  (first token after 1.4 s)
k420:~/work$
```

Figure 2-3. *The GPU name above, and the speed that Kindi measures below. The screenshot shows a live session on a Quadro K420.*

The speed in this picture comes from a real session. The `kindi-speed` script asks Kindi one question, shows the answer and prints the speed of that answer. Because it also counts the time that the network and the program need, it reads a little lower than the speed of the engine alone.

CHAPTER 3

Which models run on which cards?

Kindi 0.4.0 runs six models from three families. Four are the Ternary-Bonsai-2 models from Prism ML. All four use the same file format (PTQ1_0), and each weight has one of three values. A larger model makes fewer mistakes. It also needs more memory, and it writes more slowly on the same card.

The fifth model is NVIDIA Nemotron 3.5 Bolt E4B, from a different family (Gemma 4). Its support in Kindi is experimental. Its file is a mixed low-bit GGUF from unsloth (UD-Q3_K_XL, 4.2 GB). Kindi runs it with its own engine, with tool calls and the prompt cache. Its answers agree with llama.cpp token for token at the start of each test prompt, and then they can part on a near tie, because the two engines add their numbers in a different order.

The sixth model is AH-Nanbeige4.3-33B IQ3_XXS, our IQ3_XXS quantization of Nanbeige4.3-33B (13.1 GB). It is the largest model that Kindi runs. It runs on one card with 16 GB or more, on two cards with about 17 GB in total (two RTX 3080 or two Tesla P100 cards), and on a Quadro K6000 with part of the model in host memory (A more capable model).

Model	File size	Card	Context in our tests (tokens)	Good for
2B	429 MB	Quadro K420 1 GB	24,576	*Short decisions and small agent tasks. The fastest model, on every card.*
		Quadro K420 2 GB	65,536	
		Quadro K1200 4 GB	24,576 (fits up to 131,072)	
		The CPU	24,576	
4B	950 MB	Quadro K420 2 GB	25,088	*Agent work with more steps*
		Quadro K1200 4 GB	24,576 (fits up to 86,272)	
9B	2.0 GB	Quadro K1200 4 GB	24,576 (fits up to 47,360)	*A larger model on one small card*
27B	5.9 GB	Two Quadro K1200 cards	24,576	*Programming tasks and long documents*
		Quadro K6000 12 GB	65,536 with the installer; up to 156,672	
		GeForce GTX 1080 8 GB	65,536	
		GeForce RTX 3080 10 GB	about 85,000 (84,992) with the installer's 3080 settings; up to 103,680 without the drafter	
		Radeon 780M, 64 GB computer	24,576	
Nemotron	4.2 GB	Quadro K1200 4 GB	8,448	*Tool calls and agent work; a second model family*
		Two Quadro K1200 cards	8,448	
		GeForce GTX 1080 8 GB	24,576	
		GeForce RTX 3080 10 GB	24,576	
		Radeon 780M, 64 GB computer	24,576	

Model	File size	Card	Context in our tests (tokens)	Good for
33B	13.1 GB	Two GeForce RTX 3080 10 GB cards	16,384	*The largest model: coding work on one 24 GB card, on two cards, or on a K6000*
		Two Tesla P100 16 GB cards	up to 64K	
		Quadro K6000 12 GB	8,448	

The 9B does not fit on the 2 GB K420: its weights alone are larger than the memory of the card. Two K1200 cards hold the 27B together (Can two cards work as one?). The Radeon 780M has no memory of its own: it uses the memory of the computer, so the 27B fits in a computer with enough memory (One small box for everything). The CPU engine runs the Bonsai models only. The installer shows the models that fit your hardware (How do you start on a new computer?).

CHAPTER 4

Which card for which job?

Each card in this chapter runs Kindi in our tests. For each card, you get the memory, the power and the size. You also get the models that fit, a measured speed and the jobs that the card does well. All speeds are in tokens per second (tok/s). How fast is each card? gives every measurement and its conditions.



Quadro K420. Photo: NVIDIA data sheet.

KEPLER · 2014

NVIDIA Quadro K420

Memory	1 GB DDR3 (a 2 GB model also exists)
Power	41 W, all from the PCIe slot. No power cable.
Size	Low profile, one slot
Models	2B on the 1 GB card; 2B or 4B on the 2 GB card
Speed	1 GB card: the 2B writes **22.9 tok/s** with the fast head at 2,000 tokens. With the exact head, it writes 20.8 tok/s at 512 tokens and 15.3 at 24,000. 2 GB card: the 2B writes 21.2 tok/s and the 4B 9.8 tok/s at position 100 (`kindi bench`, build bb4008c).
Good for	Short text decisions, small agent tasks, a private assistant on an old office computer

> Many small office computers from Dell and HP take this card. Kepler cards need the NVIDIA 470 driver.



Quadro K1200. Photo: PNY.

MAXWELL · 2015

NVIDIA Quadro K1200

Memory	4 GB GDDR5
Power	45 W, all from the PCIe slot. No power cable.
Size	Low profile, one slot
Models	2B, 4B or 9B; the 27B with a second K1200
Speed	At 512 tokens: **81.0 tok/s** with the 2B, 39.2 with the 4B, 23.4 with the 9B
Good for	Agent work with the 4B, a 9B model in a small office computer

> Put the card in an x16 slot if you can. In our tests, a K1200 in an x16 slot was faster than a K1200 in an x4 slot.



Two of these cards make one engine. Photo: PNY.

TENSOR PARALLEL

Two Quadro K1200 cards

Memory	8 GB in total, 4 GB on each card
Power	90 W for the two cards, all from the slots
Size	Two low-profile slots, with space for air between the cards
Models	27B, one model on both cards
Speed	The 27B writes **12.6 tok/s** at a short context and 9.84 at 22,000 tokens
Good for	A 27B model in a small office computer

Run the computer in text mode, with no desktop. A desktop session turns on a CUDA watchdog, and the watchdog stops the kernels that exchange data between the cards.



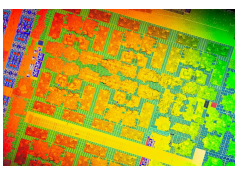
Quadro K6000. Photo: TopGear-V12, CC BY-SA 4.0.

KEPLER · 2013

NVIDIA Quadro K6000

Memory	12 GB GDDR5
Power	225 W, with two 6-pin PCIe power cables
Size	Full height, two slots
Models	27B on one card, with a context of 65,536 tokens (up to 156,672)
Speed	The 27B reads **36.2 tok/s** and writes **20.4 tok/s** at 512 tokens, about 10 times as fast as llama.cpp
Good for	The 27B with a long context on one used workstation card

Kepler cards need the NVIDIA 470 driver. The card needs a full-height case, two PCIe power cables and good airflow: it has one fan for 225 W.



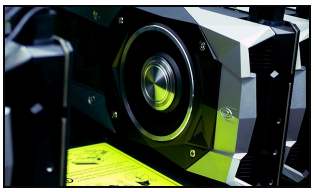
The GP100 chip of a Tesla P100. Photo: Fritzchens Fritz, Wikimedia Commons, CC0.

PASCAL · 2016 · HBM2

NVIDIA Tesla P100

Memory	16 GB HBM2, 732 GB/s
Power	250 W (PCIe card) or 300 W (SXM2 module). No fan: it needs air flow from the case.
Size	Full height, two slots (PCIe); the SXM2 module needs a server board
Models	2B, 4B, 9B and 27B on one card; the 33B on two cards
Speed	The 27B writes **36.0 tok/s** at 512 tokens, against 18.7 for llama.cpp on the same card
Good for	The cheapest way to 16 GB of fast memory: a used card costs $78 to $160

The P100 has no fan and no video output. Put it in a server, or add a blower and an adapter for the power cable. Tesla P100 16 GB gives the measurements.



GeForce GTX 1080 cards. Photo: Benedikt S. Vogler, CC BY-SA 2.0.

PASCAL · 2016

NVIDIA GeForce GTX 1080

Memory	8 GB GDDR5X
Power	180 W, with one 8-pin PCIe power cable
Size	Full height, two slots
Models	27B with a context of 65,536 tokens
Speed	The 27B writes **29.4 tok/s** at 512 tokens and 16.3 at 64,000
Good for	The 27B on one card, long documents, programming tasks

> The card needs a full-height case and a power supply with an 8-pin PCIe cable. NVIDIA recommends a 500 W power supply for the computer.



GeForce RTX 3080. Image: NVIDIA.

AMPERE · 2020 · TENSOR CORES

NVIDIA GeForce RTX 3080

Memory	10 GB GDDR6X
Power	320 W on the NVIDIA design. Many partner cards allow more; ours have a default limit of 380 W.
Size	Full height, two slots or more
Models	27B with the 2B as a drafter: about 85,000 tokens of context (84,992) with the installer's 3080 settings, up to 103,680 without the drafter
Speed	The 27B writes **63.2 tok/s** at a depth of 1K at 190 W, and 66.4 at 380 W. Nemotron writes 123 tok/s at 190 W.
Good for	Fast agent work with the 27B, very long documents

> NVIDIA lists a 750 W power supply for a computer with this card. A power cap lowers the power and the heat. What does a power cap change? shows the cost in speed.



A mini PC with Radeon 780M graphics: the Minisforum UM790 Pro. Image: Minisforum.

RDNA3 · 2023 · BUILT IN

AMD Radeon 780M

Memory	None of its own. It uses the memory of the computer: in our test computer, 64 GB of DDR5-5600.
Power	Part of the chip. The whole computer runs from one power adapter.
Size	No card: the graphics are inside the AMD Ryzen 7 7840HS, Ryzen 9 7940HS and similar chips
Models	27B, Nemotron or 2B on the graphics, and a second model on the CPU at the same time
Speed	The 27B reads **104 tok/s** and writes **10.6 tok/s** at 512 tokens
Good for	A small, quiet computer that does agent work and short text jobs at the same time

> Kindi runs on this graphics through ROCm, which is inside the Kindi image. The computer needs one kernel option, and the installer asks before it adds it. One small box for everything tells the whole story.

Will the card fit your computer?

- **Low-profile cards (K420 and K1200)** fit many small-form-factor (SFF) office computers from Dell and HP. They take all their power from the slot, so they need no power cable.
- **Two cards** need two usable slots and space for air between them.
- **Full-height cards (GTX 1080 and RTX 3080)** need a tower case, PCIe power cables and a larger power supply.
- **A mini PC without a PCIe slot cannot take a card.** If its chip has Radeon 780M graphics, Kindi runs on those graphics. For other mini PCs, a used SFF or tower office computer with a card is often the practical base.

Before you buy a card, check your computer. Look at the bracket height, the space between the slots, the power supply, the airflow, the BIOS and the driver.

What about Tensor Cores and FP64?

The RTX 3080 has third-generation Tensor Cores. These units do fast matrix arithmetic with small numbers, such as 8-bit integers. Kindi uses them on the 3080, for the 27B and for the prompts of Nemotron. The K420, the K1200 and the GTX 1080 have no Tensor Cores. On these cards, Kindi uses its own CUDA kernels. The

Radeon 780M has similar matrix instructions (WMMA), and Kindi uses them for the prompts of Nemotron.

On GeForce RTX 30 cards, NVIDIA runs 16-bit matrix arithmetic (FP16 and BF16) at half rate. A third-party kernel module, gpu-unlock, removes this limit at boot. Kindi does not include it. The installer tells you which of your cards are not unlocked and offers to install the module, and it asks you at the keyboard before it does.

FP64 (double precision) is a different measure. Some old compute cards are strong in FP64. The Tesla K80 has a peak of about 1.87 TFLOPS, the Quadro K6000 about 1.70 and the RTX 3080 about 0.465. Scientific programs use FP64. Kindi uses smaller number formats, so a high FP64 number does not make a card fast for Kindi.

CHAPTER 5

How fast is each card?

This chapter gives the full performance matrix. For each card, you get a chart, a table and the conditions of the test. In short:

- **On the Kepler and Maxwell cards (K420 and K1200),** Kindi reads and writes several times faster than llama.cpp at each context with the Bonsai models. With Nemotron on one K1200, Kindi reads about 4 times and writes about 1.3 times as fast.
- **On the Quadro K6000,** Kindi reads the 27B 4 to 5 times as fast as llama.cpp and writes it 7 to 10 times as fast. At 24,000 tokens, llama.cpp did not finish the prompt in one hour.
- **On the GTX 1080,** Kindi is faster with the 27B at each context that the two engines share, and it reaches 64,000 tokens, where llama.cpp runs out of memory. With Nemotron, Kindi reads prompts of 8,000 tokens and more faster, llama.cpp reads a 512-token prompt faster, and llama.cpp writes 2% to 7% faster.
- **On the RTX 3080,** the two engines read at about the same speed. Kindi writes faster, and the difference grows at long context. With the 2B drafter, Kindi is the fastest setup that we measured on agent prompts. With Nemotron, Kindi writes about 1.4 times as fast.
- **On two K1200 cards,** Kindi writes 12.6 tok/s with the 27B and llama.cpp writes 1.69 tok/s with the same model file. Can two cards work as one? gives the details, also for Nemotron.
- **On the Radeon 780M,** Kindi reads the 27B 1.5 to 1.7 times as fast as llama.cpp at each context, and it writes 1.4 times as fast at 24,000 tokens. With Nemotron (experimental), llama.cpp is faster.
- **On the CPU,** Kindi's CPU engine reads 1.3 to 1.5 times as fast as llama.cpp and writes 1.6 to 2.7 times as fast, with the 2B and the 4B on two common CPUs.

- **On the Tesla P100,** Kindi writes the 27B 1.9 to 2.7 times as fast as llama.cpp, and it reads prompts 12% to 14% faster. Tesla P100 16 GB gives the details.
- **With AH-Nanbeige4.3-33B on two RTX 3080 cards,** Kindi writes 19% to 41% faster than llama.cpp, and it reads prompts as fast or faster. A more capable model gives the details.

What do the numbers mean?

- **Read speed** is the speed at which the engine reads your prompt. An agent sends a long prompt, so the read speed decides how long you wait for the first word of the answer.
- **Write speed** is the speed at which the engine writes the answer, one token after the other.
- **Context** is the number of tokens that the model can see at one time: the prompt, the conversation and the answer. Both speeds fall as the context grows, because the model must look back at more tokens. So each row gives the context of the measurement. "Depth" is the position in the context where the engine writes.
- **Exact head and fast head.** The head is the last step of the model: it selects the next token. The exact head gives the same tokens as llama.cpp. The fast head is a little faster, and its output can be a little different. All the comparisons in this chapter use the exact head.
- **Greedy and sampled.** Greedy decoding always takes the most probable token. Sampled decoding adds some chance, and pi uses it.
- **Speculative decoding.** A small model (the drafter) guesses the next few tokens. The large model then checks all of them in one pass. When the guesses are correct, one pass gives several tokens. With greedy decoding, the output is the same as the output without the drafter. On NVIDIA RTX 30 cards and later, Kindi 0.4.0 uses the 2B as a drafter for the 27B by default (How does a draft model make the 27B faster?).

How did we compare the engines?

We use one rule for each comparison: the same card, the same model file and the same context for both engines. We give the read speed and the write speed of each engine at each context. When only one engine reaches a context, the row says so:

"llama.cpp: out of memory" or "Kindi: does not fit". When a run crashed, the row says "crashed". A crash is a failure of the run, and it does not tell you a speed.

We compare Kindi with llama.cpp, the engine that Kindi's output matches. We round each speed to three significant figures, and we give speeds of 100 tok/s and more as whole numbers. Each table names its build and date.

Quadro K420 1 GB

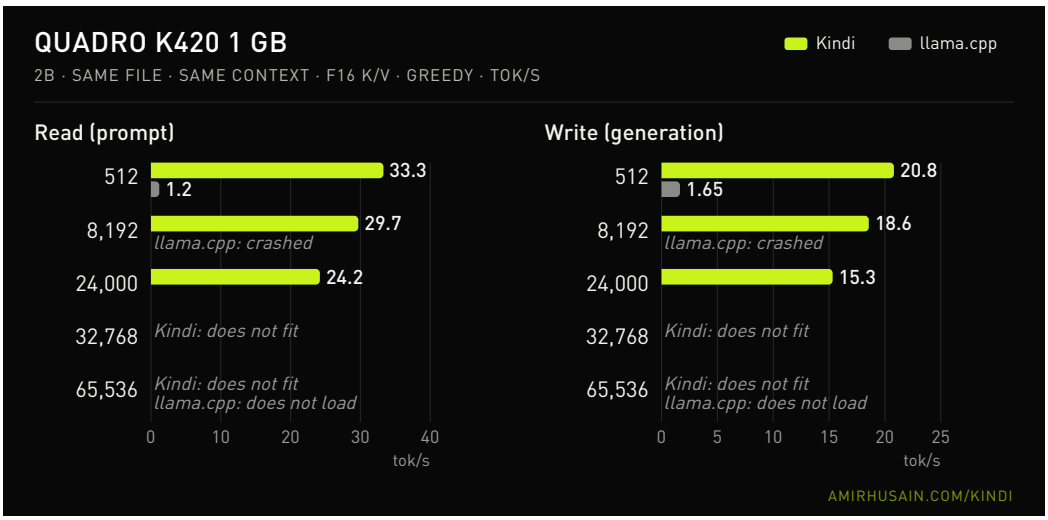


Figure 5-1. *Read and write speed on the Quadro K420 1 GB with the 2B. Kindi is lime and llama.cpp is grey. At 8,192 tokens, the llama.cpp run crashed; at 32,768 tokens, Kindi does not fit.*

Context (tokens)	Kindi read	llama.cpp read	Kindi write	llama.cpp write
512	**33.3**	1.2	**20.8**	1.65
8,192	**29.7**	*llama.cpp: crashed*	**18.6**	*llama.cpp: crashed*
24,000	**24.2**	–	**15.3**	–
32,768	*Kindi: does not fit*	*llama.cpp: loads*	*Kindi: does not fit*	*llama.cpp: loads*
65,536	*Kindi: does not fit*	*llama.cpp: does not load*	*Kindi: does not fit*	*llama.cpp: does not load*

Conditions. Quadro K420 1 GB. Model Ternary-Bonsai-2-2B-PTQ1_0.gguf (sha256 36061ad4…) for both engines. f16 K/V cache for both. The same 24,576 token ids for both engines, greedy. One pass: the read time is the sum of the segments up to each context, and the engine writes 32 tokens at each depth. llama.cpp: the prebuilt Vulkan release, flash attention on. Kindi: Kindi 0.2.0, exact head, prompts in chunks of 128 tokens. llama.cpp: 2026-10-21. Kindi: 2026-10-23.

At 512 tokens, Kindi reads 33.3 tok/s and writes 20.8 tok/s. llama.cpp reads 1.2 tok/s and writes 1.65 tok/s. At 8,192 tokens, the llama.cpp run stopped after 17 minutes with a Vulkan driver error (ErrorDeviceLost). The largest context for Kindi on this card is 24,576 tokens, so Kindi does not fit 32,768. llama.cpp loads a context of 32,768 tokens, and it does not load 65,536. After the crash at 8,192 tokens, the table has no deeper llama.cpp rows.

Kindi 0.1.0 read the same 512-token prompt at 25.1 tok/s, one token at a time. Kindi 0.2.0 reads the prompt in chunks, and one pass of the weights serves all the tokens of a chunk.

Quadro K1200 4 GB



Figure 5-2. *Read and write speed on one Quadro K1200 with the 2B, the 4B and the 9B. Kindi is lime and llama.cpp is grey. The llama.cpp run of the 9B did not finish the 24,000-token prompt in its one-hour limit.*

Context (tokens)	Model	Kindi read	llama.cpp read	Kindi write	llama.cpp write
512	2B	**195**	26.7	**81.0**	20.8
8,192	2B	**179**	28.1	**73.3**	21.9
24,000	2B	**152**	26.5	**58.8**	18.6
512	4B	**77.4**	11.1	**39.2**	8.7
8,192	4B	**70.2**	10.8	**34.3**	8.1
24,000	4B	**58.9**	10.2	**26.4**	6.9
512	9B	**41.7**	5.8	**23.4**	5.9
8,192	9B	**39.2**	5.7	**21.5**	5.6
24,000	9B	**35.6**	*llama.cpp: not finished*	**18.1**	*not finished*

> **Conditions.** One Quadro K1200 4 GB in a PCIe x16 slot. Models Ternary-Bonsai-2-2B, 4B and 9B-PTQ1_0.gguf, the same files for both engines. f16 K/V cache for both. The same token ids, greedy, the same method as the K420. llama.cpp: Prism fork, Vulkan build, flash attention on, a one-hour limit for each model. Kindi: Kindi 0.2.0, exact head, prompts in chunks. 2026-10-22.

At 512 tokens, Kindi writes 81.0 tok/s with the 2B, 39.2 with the 4B and 23.4 with the 9B. llama.cpp writes 20.8, 8.7 and 5.9. Kindi also reads several times faster at each context. The llama.cpp run of the 9B read 8,192 tokens of the 24,000-token prompt at 5.7 tok/s, and then its one-hour limit ran out.

Kindi fits the context to the free memory of the card. This table gives the largest context of each model:

Model	Kindi: largest context on one K1200	llama.cpp: load test
2B	131,072 (all that we asked for)	loads at 65,536 (larger not tried)
4B	86,272	loads at 65,536 (larger not tried)
9B	47,360	loads at 65,536 (Kindi does not fit there)

With the 9B, llama.cpp loads a context of 65,536 tokens, and Kindi does not fit there. Kindi's largest 9B context on this card is 47,360 tokens.

Nemotron also fits on one K1200:

Context (tokens)	Kindi read	llama.cpp read	Kindi write	llama.cpp write
512	**66.8**	17.6	**10.4**	8.27
8,000	**73.9**	12.0	**9.07**	7.67

> **Conditions.** The same card. Model NVIDIA-Nemotron-3.5-Bolt-E4B-UD-Q3_K_XL.gguf (sha256 3f8032b9…) for both engines. f16 K/V for both, the same token ids, greedy, one pass, 32 tokens of write at each depth. Context 8,448 (all that fits on one card). llama.cpp: upstream Vulkan build 1b29621. Kindi: Kindi 0.3.0 development build (5350b87 with the server fix; the same engine kernels as the release). 2026-10-24.

With Nemotron on one K1200, Kindi reads a 512-token prompt at 66.8 tok/s, and llama.cpp at 17.6. Kindi writes 10.4 tok/s, and llama.cpp writes 8.27.

GeForce GTX 1080 8 GB

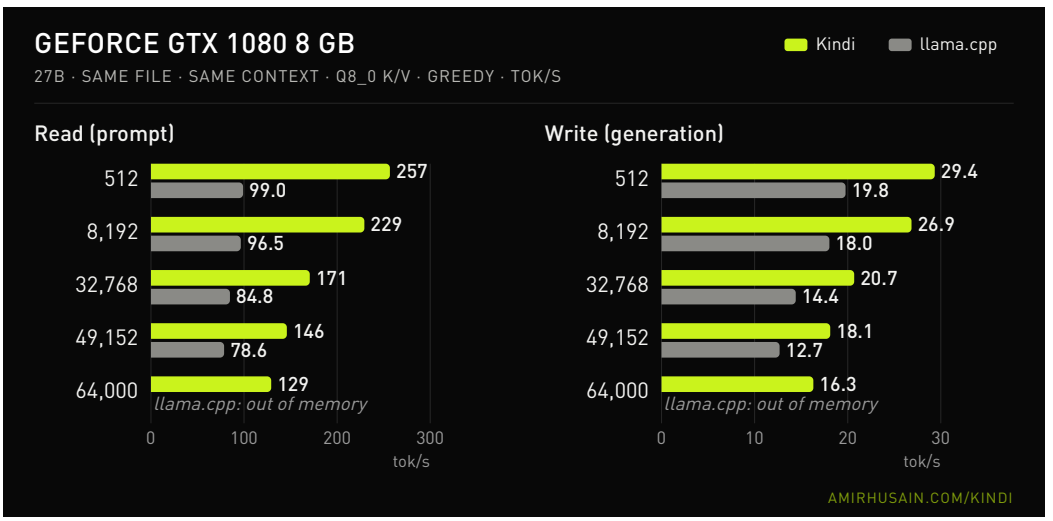


Figure 5-3. *Read and write speed on the GeForce GTX 1080 with the 27B. Kindi is lime and llama.cpp is grey. At 64,000 tokens, llama.cpp runs out of memory.*

Context (tokens)	Kindi read	llama.cpp read	Kindi write	llama.cpp write
512	**257**	99.0	**29.4**	19.8
8,192	**229**	96.5	**26.9**	18.0
32,768	**171**	84.8	**20.7**	14.4
49,152	**146**	78.6	**18.1**	12.7
64,000	**129**	*llama.cpp: out of memory*	**16.3**	*llama.cpp: out of memory*

Conditions. GeForce GTX 1080 8 GB, one card for both engines. Model Ternary-Bonsai-2-27B-PTQ1_0.gguf (sha256 53107f53…) for both. q8_0 K/V cache for both. The 22,000-token reference prompt, repeated, as the same token ids for both engines, greedy, the same method as the K420. llama.cpp: Prism fork, CUDA build for sm_61, flash attention on, context 49,280; at 65,600 it does not load (out of memory). Kindi: Kindi 0.2.0, one-card engine, 5-trit weights, int8 prompt path, exact head, context 65,536. llama.cpp: 2026-10-21. Kindi: 2026-10-22.

At 512 tokens, Kindi reads 257 tok/s and llama.cpp reads 99.0. Kindi writes 29.4 tok/s and llama.cpp writes 19.8. Kindi is faster at each context that the two engines share. At 64,000 tokens, llama.cpp runs out of memory, and Kindi writes 15.4 tok/s.

Two methods make this possible. Kindi stores the weights of the 27B in a compact 5-trit format, so the model and a context of 65,536 tokens fit in 8 GB. And on this card, Kindi reads the prompt with 8-bit integer (int8) arithmetic, the default on sm_61 and later cards.

With Nemotron on the same card:

Context (tokens)	Kindi read	llama.cpp read	Kindi write	llama.cpp write
512	**743**	1,088	**55.1**	56.4
8,192	**922**	897	**51.1**	53.1
16,384	**838**	731	**47.8**	50.4
24,000	**747**	627	**45.2**	48.2

Conditions. GeForce GTX 1080 8 GB, one card for both engines. Model NVIDIA-Nemotron-3.5-Bolt-E4B-UD-Q3_K_XL.gguf (sha256 3f8032b9…) for both. f16 K/V for both, the same token ids, greedy, the same method as the 27B, and both engines warmed first: the first request of llama.cpp includes an 11-second warm-up. llama.cpp: CUDA build b11804 (c7fc509), flash attention on. Kindi: Kindi 0.3.0 development build (849840a). 2026-10-25.

From 8,192 tokens on, Kindi reads faster: 922 against 897 tok/s at 8,192, and 747 against 627 at 24,000. With a 512-token prompt, llama.cpp reads faster: 1,088 against 743 tok/s. llama.cpp also writes faster at each depth, by 2% to 7%: 56.4 against 55.1 tok/s at 512 tokens, and 48.2 against 45.2 at 24,000.

Tesla P100 16 GB

The Tesla P100 is a supported card in Kindi 0.4.0. It is a data center card from 2016 with 16 GB of HBM2 memory at 732 GB/s, and a used card costs $78 to $160. This is

a good deal: Kindi writes the 27B 1.9 to 2.7 times as fast as llama.cpp on the same P100, and a computer with two P100 cards runs the 33B.

Model and context	Kindi read	llama.cpp read	Kindi write	llama.cpp write
27B, 512 tokens	**168**	146	**36.0**	18.7
27B, 8,192 tokens	**160**	143	**34.6**	17.7

> **Conditions.** Box A: one Tesla P100-SXM2-16GB (NVLink, 300 W) of a rented machine, 32 CPU threads, driver 535.230.02. Kindi with the fp16 prompt path on; llama.cpp (Prism fork c96baf4, CUDA 12.8, `-ngl 99 -fa 1`, 3 runs). The 27B file, and q8_0 K/V in both engines. Read is a prompt from an empty cache; write is the speed at that depth. `kindi verify`: 4 of 4 prompts token-exact.

With the fp16 prompt path, Kindi reads prompts 14% faster than llama.cpp at 512 tokens and 12% faster at 8,192. The path is on by default for this card (`KINDI_P100_FP16_PROMPT=0` turns it off). Without it, Kindi reads at about two thirds of the speed of llama.cpp. On two cards in box A, Kindi writes 39.7 tok/s at 512 tokens.

A second rented machine (box B, PCIe cards at 200 W) gave these write speeds for the 27B on two cards, and on one card:

Context (tokens), two cards	Kindi	llama.cpp	Times as fast
512	**40.4**	14.9	2.71×
8,192	**39.5**	15.9	2.48×
30K	**34.9**	13.6	2.57×

Context (tokens), one card	Kindi	llama.cpp	Times as fast
512	**29.6**	14.4	2.06×
8,192	**26.8**	13.3	2.02×

With the 33B on two cards, Kindi writes about 2.2 times as fast as llama.cpp (box B):

Context (tokens)	Kindi	llama.cpp	Times as fast
512	**21.0**	9.6	2.19×
8,192	**19.6**	8.9	2.20×

Kindi writes the 33B at 15.7 tok/s at 32K tokens and 12.3 tok/s at 64K tokens.

Two P100 cards (box B) hold the full 262,144 tokens of context of the 27B, with 8.6 GB free on each card. Kindi reads a prompt of 262,016 tokens at 67.6 tok/s (3,877 s) and writes 15.8 tok/s at that depth.

Speculative decoding is off on the P100. On this card, a pass over several tokens costs about as much as the same number of single steps, so a draft cannot save time. `kindi verify` passes on one card for the 2B, the 4B and the 27B, and on two cards for the 33B.

GeForce RTX 3080 10 GB

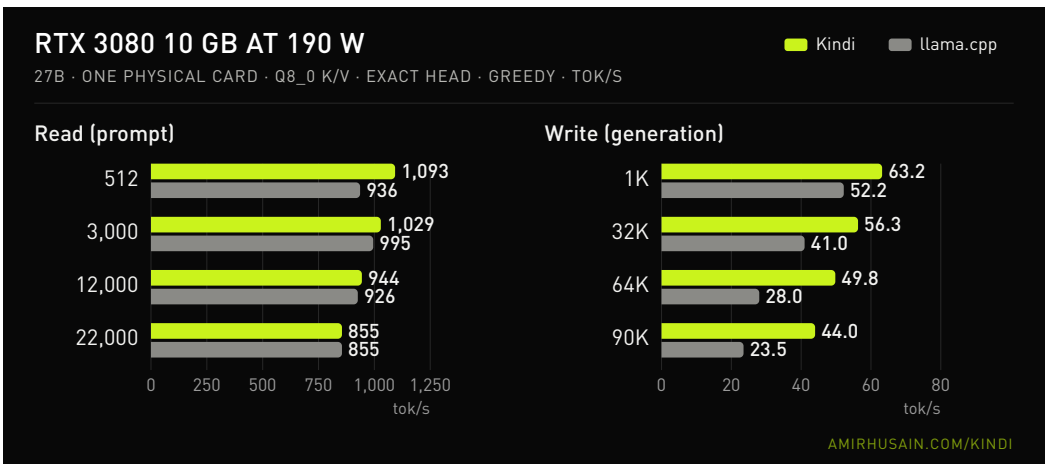


Figure 5-4. *Read and write speed on one RTX 3080 at a 190 W power cap with the 27B, exact head, Kindi 0.2.0. Kindi is lime and llama.cpp is grey.*

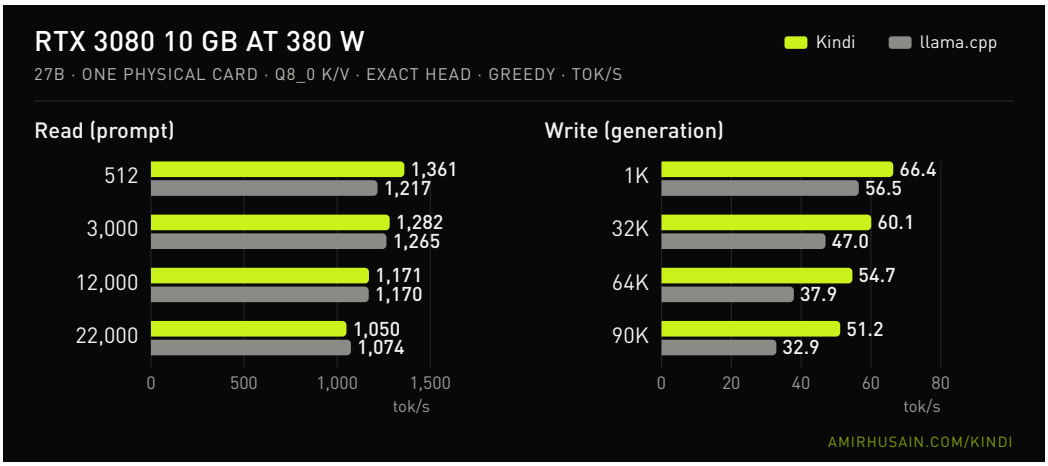


Figure 5-5. *The same card and the same tests at a 380 W power cap, Kindi dev build release-tcspec6 (2026-10-22).*

	190 W power cap		380 W power cap	
Read: prompt (tokens)	**Kindi**	**llama.cpp**	**Kindi**	**llama.cpp**
512	**1,093**	936	**1,361**	1,217
3,000	**1,029**	995	**1,282**	1,265
12,000	**944**	926	**1,171**	1,170
22,000	**855**	855	**1,050**	1,074

	190 W power cap		380 W power cap	
Write: depth (tokens)	**Kindi**	**llama.cpp**	**Kindi**	**llama.cpp**
1K	**63.2**	52.2	**66.4**	56.5
32K	**56.3**	41.0	**60.1**	47.0
64K	**49.8**	28.0	**54.7**	37.9
90K	**44.0**	23.5	**51.2**	32.9

Conditions. One GeForce RTX 3080 10 GB for both engines, a card of our RTX 3080 cluster on a PCIe x1 riser. Model Ternary-Bonsai-2-27B-PTQ1_0.gguf (sha256 53107f53…) for both. q8_0 K/V cache for both, flash attention on. Read: a prompt of N tokens from an empty cache. Write: 32 greedy tokens after a prompt of D tokens. The riser slows the model load, and it does not change these measurements. No CPU limit (four cores). llama.cpp: Prism fork at commit 1180918, CUDA 12.4 build for sm_86. Kindi: exact head, tensor-core engine. 190 W: Kindi 0.2.0 (engine commit 2221315). 380 W: dev build release-tcspec6 (2026-10-22). The largest context in the 190 W run: Kindi 103,680 tokens, llama.cpp 90,000. 2026-10-22.

At 190 W, Kindi writes 63.2 tok/s at a depth of 1K, and llama.cpp writes 52.2. The difference grows with the depth. At 90K, Kindi writes 44.0 tok/s, and llama.cpp writes 23.5.

The read speeds are close. At 190 W, Kindi reads faster up to 12,000 tokens (1,093 against 936 tok/s at 512), and the two engines read at the same speed at 22,000 (855 tok/s). At 380 W, Kindi reads faster up to 12,000 tokens, and llama.cpp reads faster at 22,000.

How much CPU does each engine use?

Kindi needs very little time on the host CPU. At 190 W, Kindi 0.2.0 used 0.41 to 1.02 ms of CPU time for each token. In build release-tcspec6, llama.cpp used about 13 ms. This table, from release-tcspec6, gives the speeds with four CPU cores and with one core:

	Kindi		llama.cpp	
Measure	4 cores	1 core	4 cores	1 core
190 W, read at 512	1,069	1,092	960	910
190 W, read at 22,000	838	858	853	828
190 W, write at 1K	62.4	63.9	52.9	52.0
190 W, write at 90K	43.6	45.9	24.7	23.5
380 W, read at 512	1,361	1,360	1,217	1,208
380 W, read at 22,000	1,050	1,050	1,074	1,068
380 W, write at 1K	66.4	66.2	56.5	56.5
380 W, write at 90K	51.2	53.0	32.9	33.0

What does the 2B drafter add?

These are the speeds on real agent prompts: 30 greedy and 20 sampled requests, with prompts of 1.1K to 10.5K tokens and 128 tokens out.

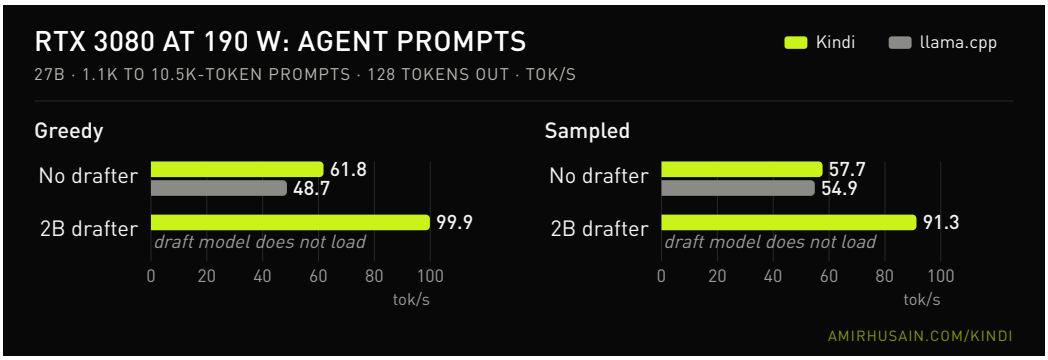


Figure 5-6. *Write speed on agent prompts on one RTX 3080 at 190 W, with the 27B, Kindi 0.2.0. Kindi is lime and llama.cpp is grey. llama.cpp cannot load the 2B as a draft model for the 27B.*

	190 W		380 W	
Engine	greedy	sampled	greedy	sampled
llama.cpp	48.7	54.9	53.2	60.8
Kindi, exact	**61.8**	**57.7**	**66.9**	**58.6**
Kindi + 2B drafter	**99.9**	**91.3**	**108**	**90.9**
llama.cpp + 2B drafter	*draft model does not load (both caps)*			

> **Conditions.** The same card, files and builds as the tables above (190 W: Kindi 0.2.0; 380 W: dev build release-tcspec6, 2026-10-22). llama-server with flash attention, q8_0 K/V, context 81,920, one slot, no host prompt cache. Kindi with the 2B drafter: three draft tokens in each verify pass. 2026-10-22.

With the 2B drafter at 190 W, Kindi writes 99.9 tok/s greedy and 91.3 tok/s sampled: 1.62 and 1.58 times its exact speed. The greedy output with the drafter was identical to the exact output in all 30 requests. At 380 W, Kindi writes 108 tok/s greedy and 90.9 tok/s sampled. llama.cpp at commit 1180918 cannot load the 2B as a draft model for the 27B, because the two files carry different Prism metadata. So llama.cpp has no drafter row.

Nemotron on the RTX 3080

Hardware	Kindi read	llama.cpp read	Kindi write	llama.cpp write
RTX 3080, 190 W	**2,614**	2,630	**123**	88.0

> **Conditions.** One RTX 3080 of the cluster at 190 W, with the gpu-unlock module loaded. Model NVIDIA-Nemotron-3.5-Bolt-E4B-UD-Q3_K_XL.gguf (sha256 3f8032b9…) for both engines. The same chat request for both, the mean of the second and third runs. Kindi: Kindi 0.3.0 development build (b340cbe). 2026-10-24.

The two engines read at about the same speed: 2,614 tok/s for Kindi and 2,630 for llama.cpp. Kindi writes 123 tok/s, and llama.cpp writes 88.0. Kindi reads the prompts of Nemotron with the Tensor Cores, and it writes each token with one recorded sequence of GPU work (a CUDA graph), so the CPU does almost nothing between tokens.

RTX 20, 30, 40 and 50 series

Kindi 0.4.0 supports all four series. We measured speed on the RTX 3080 and the RTX 3060 (above and in the chapter about the draft model). With the 2B drafter, the 27B writes 1.46 to 1.82 times as fast as without it on both cards, with the same output. A user with an RTX 4060 Laptop GPU (8 GB) ran Kindi 0.3.0 and reported that the 27B writes 28.0 tok/s at 100 tokens and 25.2 tok/s at 24,000 tokens, and that `kindi verify` passes. For the other cards of these series, Kindi is supported and `kindi verify` passes. We give no speed numbers for them.

Radeon 780M

The Radeon 780M is the graphics unit inside the AMD Ryzen 9 7940HS of our test computer. It has no memory of its own, so it reads the weights from the DDR5 memory of the computer, at about 83 GB/s. That memory speed sets the write speed of each model.

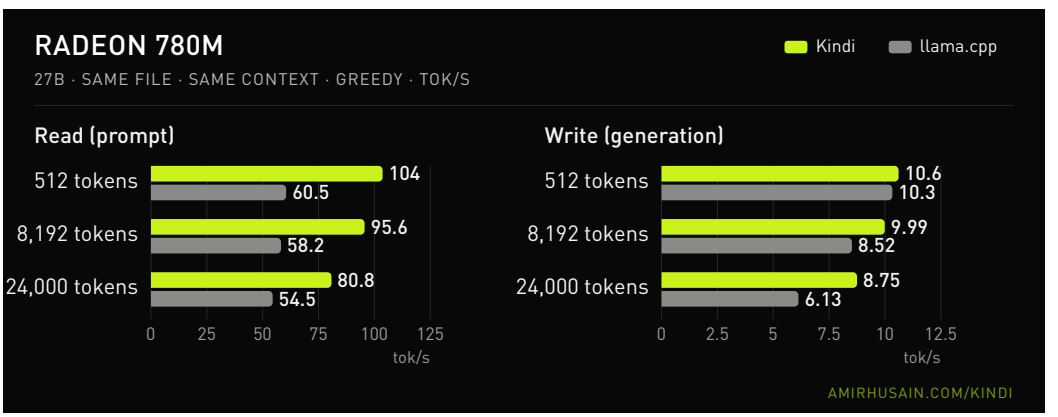


Figure 5-7. *Read and write speed on the Radeon 780M with the 27B. Kindi is lime and llama.cpp is grey.*

Model and context	Kindi read	llama.cpp read	Kindi write	llama.cpp write
27B, 512 tokens	**104**	60.5	**10.6**	10.3
27B, 8,192 tokens	**95.6**	58.2	**9.99**	8.52
27B, 24,000 tokens	**80.8**	54.5	**8.75**	6.13
Nemotron, 512 tokens	**186**	670	**19.1**	24.0
Nemotron, 8,192 tokens	**392**	599	**20.0**	21.4
Nemotron, 24,000 tokens	**239**	477	**17.0**	19.1

> **Conditions.** Minisforum UM790 Pro, AMD Ryzen 9 7940HS with Radeon 780M, 64 GB DDR5-5600, Linux kernel 6.8 with `amdgpu.cwsr_enable=0`. The same model files for both engines. Kindi: Kindi 0.3.0 development build (b5230a3), `HSA_OVERRIDE_GFX_VERSION=11.0.2`. The same token ids for both engines, greedy, one pass, 32 tokens of write at each depth, both engines warmed first. q8_0 K/V for the 27B, f16 for Nemotron. llama.cpp: Prism fork, ROCm build, for the 27B; upstream b10700, ROCm build, for Nemotron. 2026-10-25.

The 27B is the main result on this chip. Kindi reads it 1.5 to 1.7 times as fast as llama.cpp at each context: 104 against 60.5 tok/s with a 512-token prompt, and 80.8 against 54.5 at 24,000. Kindi writes 10.6 tok/s at 512 tokens, about the same as llama.cpp, and 8.75 against 6.13 at 24,000. So the advantage grows as the context grows.

Nemotron is experimental on this chip. Here llama.cpp reads it 1.5 to 3.6 times as fast, and writes it about 10% to 25% faster. We are still optimizing Kindi's Nemotron engine.

Quadro K6000

The Quadro K6000 is the largest card of the Kepler generation, from 2013: 12 GB of memory on one card. It is the same generation as the K420, but it has 15 times the compute units and 10 times the memory speed. On this card, Kindi runs the 27B on one card, as on the GTX 1080.

Model and context	Kindi read	llama.cpp read	Kindi write	llama.cpp write
27B, 512 tokens	**36.2**	7.0	**20.4**	2.07
27B, 8,192 tokens	**30.2**	6.9	**16.4**	2.15
27B, 24,000 tokens	**26.2**	*llama.cpp: did not finish in 1 hour*	**13.1**	*llama.cpp: did not finish in 1 hour*
Nemotron, 512 tokens	**188**	90.7	**20.1**	10.7
Nemotron, 8,192 tokens	**182**	33.8	**19.1**	9.8
Nemotron, 24,000 tokens	**164**	15.6	**17.4**	8.56

Conditions. Quadro K6000 12 GB (Kepler GK110, sm_35) passed through to a KVM virtual machine on a dual Xeon E5-2673 v4 host, NVIDIA driver 470.256.02. The same model files and the same token ids for both engines, greedy, 32 tokens of write at each depth, both engines warmed first. Kindi 0.3.0 development build (597d2d8) in the CUDA 10.2 image, q8_0 K/V for the 27B, f16 for Nemotron. llama.cpp: Vulkan builds (the Prism fork for the 27B, upstream 1b29621 for Nemotron), `-ngl 99 -fa on`. 2026-10-25.

With the 27B, Kindi reads 4 to 5 times as fast as llama.cpp and writes 7 to 10 times as fast: 20.4 against 2.07 tok/s at 512 tokens. At 24,000 tokens, llama.cpp did not finish the prompt in one hour; Kindi read it in about 15 minutes and then wrote 13.1 tok/s. Both engines give the same tokens (`kindi verify`, 4 of 4). Nemotron also runs on this card, and Kindi writes it about twice as fast as llama.cpp.

We also tried Kindi's tuning settings on this card: the chunk size, the tile, the split, blocking sync and the batch. None of them changed the speed by more than 3%, so the installer uses the same settings as on the other cards.

The CPU

Kindi also runs the Bonsai models on the CPU, with no graphics at all. We compared it with llama.cpp on the CPUs of two common computers, with the same model files and the same number of threads.

CPU and model	Kindi read	llama.cpp read	Kindi write	llama.cpp write
Ryzen 9 7940HS, 2B	**90.4**	60.5	**64.8**	23.6
Ryzen 9 7940HS, 4B	**37.8**	25.7	**29.1**	16.5
Core i7-8700, 2B	**42.5**	33.5	**28.9**	18.1
Core i7-8700, 4B	**17.0**	13.0	**12.6**	7.75

Conditions. Rainstorm's Intel Core i7-8700 (6 cores, 12 threads) and the AMD Ryzen 9 7940HS of the 780M computer (8 cores, 16 threads), with the graphics idle. The same model files for both engines. Read: a prompt of about 2,000 tokens with no cache reuse. Write: 256 tokens. The median of 3 runs, and each engine at its best number of threads. llama.cpp: Prism fork, CPU build. Kindi: Kindi 0.3.0 development builds (849840a on the 7940HS; 5350b87 with the server fix on the i7-8700). 2026-10-25.

On the 7940HS, Kindi's CPU engine writes the 2B at 64.8 tok/s, and llama.cpp at 23.6: 2.7 times as fast. It reads the 2B prompt at 90.4 tok/s, and llama.cpp at 60.5. On rainstorm's i7-8700, a CPU from 2017, Kindi writes the 2B at 28.9 tok/s, and llama.cpp at 18.1.

Kindi's CPU engine reads a prompt one token at a time, and it still reads faster than llama.cpp's batched prompt path. A 2B that writes 30 to 65 tok/s on the CPU is fast enough for short text jobs: a spam check or an escalation decision takes a few seconds (Fast decisions on short text).

How well does each card do agent work?

These are the results of the agent tasks in the Kindi benchmark (kbench), with pi. The benchmark has 24 terminal tasks, 32 programming and repair tasks, and 9 chains of small prompts. Each task gets one pi session and a hidden test.

Card	Model	Terminal	Programming and repair	Chains	Time per terminal task
Quadro K420 1 GB	2B	**13 / 24**	**7 / 32**	1 / 9	96 s
Quadro K1200	2B	**16 / 24**	**6 / 32**	0 / 9	25 s
Quadro K1200	4B	**21 / 24**	**23 / 32**	6 / 9	24 s
2× K1200, one 9B server on each card	9B	**22 / 24 †**	**26 / 32**	–	37 s and 176 s
GeForce GTX 1080	27B	**24 / 24**	**32 / 32**	7 / 9	28 s

Conditions. kbench 0.1.x with pi, thinking on, 4,096 output tokens, one run of each task, 1,800 s for each task, exact head. † The terminal score of the 9B is a composite: the first run used the fast head, and we ran the 6 failed tasks again with the exact head. The 9B ran on two servers, one on each K1200, so the time column gives one time for each server. The K420 terminal tasks ran on Kindi 0.1.0.

The larger models do much better. On one K1200, the 4B passed 23 of the 32 programming tasks, and the 2B passed 6. On the GTX 1080, the 27B passed all 24 terminal tasks and all 32 programming tasks.

CHAPTER 6

How does a draft model make the 27B faster?

To write one token, the 27B reads all of its weights from the memory of the card. On a card with Tensor Cores, a pass over two, three or four tokens takes little more time than a pass over one token. Kindi 0.4.0 uses this. The 2B, a much smaller model, guesses the next tokens. Then the 27B checks all of the guesses in one pass. Kindi keeps each guess that is the same as the token of the 27B, and it adds one token of the 27B itself. When the guesses are correct, one pass of the 27B gives several tokens.

The 27B makes every decision. A wrong guess costs a little time, and it does not change the text. With greedy decoding, the output is the same as without the drafter, token for token. With sampled decoding, Kindi checks each guess against a sample from the 27B, so the text comes from the same distribution as without the drafter.

The installer turns this on for the 27B on NVIDIA cards of compute capability 8.0 and later: the RTX 30, 40 and 50 series. You do not have to set anything.

How fast is it?

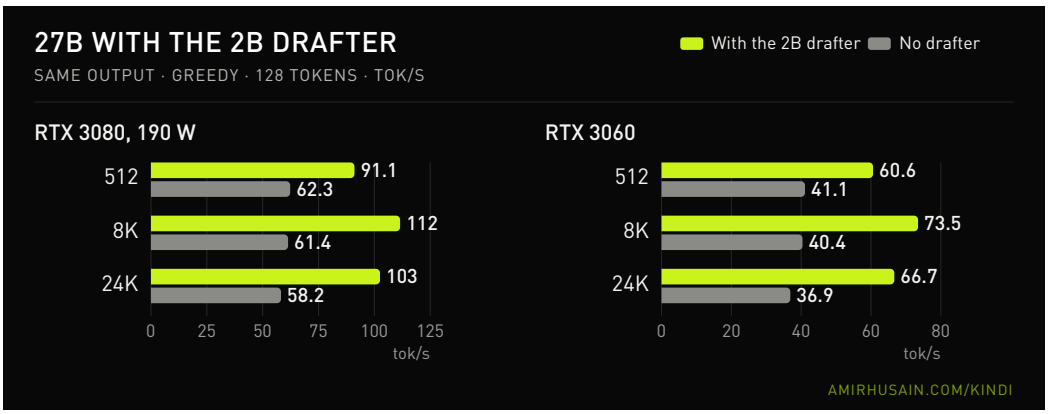


Figure 6-1. *Write speed of the 27B with and without the 2B drafter on an RTX 3080 at 190 W and on an RTX 3060. The output is the same in each row. With the drafter is lime, and without it is grey.*

Card	Context (tokens)	No drafter	With the 2B drafter	Faster
RTX 3080, 190 W	512	62.3	**91.1**	1.46×
	8K	61.4	**112**	1.82×
	24K	58.2	**103**	1.76×
RTX 3060	512	41.1	**60.6**	1.47×
	8K	40.4	**73.5**	1.82×
	24K	36.9	**66.7**	1.81×

> **Conditions.** Model Ternary-Bonsai-2-27B-PTQ1_0.gguf (sha256 53107f53…), with Ternary-Bonsai-2-2B as the drafter. Kindi 0.4.0 development build (1fcf543), tensor-core engine, exact head. The same held-out text on each card, greedy, 128 tokens after a prompt of the given length. The drafter guesses 2 tokens for each pass below 1,000 tokens of context, and 3 tokens from 1,000 tokens. These are the settings that the installer uses. RTX 3080: a card of our RTX 3080 cluster at 190 W, PCIe x1, a host with 4 CPUs. RTX 3060: PCIe x4. In every row, the tokens were the same as without the drafter. 2026-10-30.

On the RTX 3080 at 190 W, the 27B writes 91.1 tok/s with the drafter at 512 tokens, against 62.3 without it. At 8K, it writes 111.5 tok/s, against 61.4. On the RTX 3060, it writes 60.6 against 41.1 at 512 tokens, and 73.5 against 40.4 at 8K.

Does it change the agent results?

No. We ran the 65 tasks of the Kindi benchmark with pi on the 27B, on one RTX 3080 at 190 W, with the settings of the installer. The 27B passed 64 of the 65 tasks. Kindi 0.3.0 passed the same 64 tasks with the same model file and settings. The one task that failed was the same task in both runs. During the agent turns, the 27B wrote about 93 tok/s.

> **Conditions.** kbench 0.1.1 with pi 1.3.0, thinking on, one run of each task, 1,200 s for each task, context 16,384. Kindi 0.4.0 development build (59468a9). 2026-10-30.

Is the 2B faster too?

Yes, a little. The 2B has no smaller model to guess for it. In place of a drafter, it uses a small extra layer, an MTP head, that predicts the token after the next one. The installer downloads the head from Hugging Face and checks its sha256.

Card	Context (tokens)	No head	With the head	Faster
RTX 3080, 190 W	512	371	**409**	1.10×
	8K	369	**396**	1.07×
RTX 3060	512	317	**351**	1.11×
	8K	314	**340**	1.08×

> **Conditions.** Model Ternary-Bonsai-2-2B-PTQ1_0.gguf with its head (`kindi-mtp-2b-v1.npz`). The same cards, build and text as the table above, greedy, 512 tokens, the median of 3 runs. The head guesses 2 tokens for each pass. 2026-10-30.

The head makes the 2B about 1.1 times as fast, up to 8K tokens of context. Above 8K, the head gave no gain, so Kindi turns it off there. The head works with greedy decoding only.

Which cards use it?

Speculation runs only on the tensor-core path of Kindi: NVIDIA cards of compute capability 8.0 and later. On the other cards, a pass over two tokens costs almost as much as two single steps: 1.80 steps on the Quadro K1200 with the 2B, and 1.82 on the GTX 1080 with the 27B. There, a guess cannot save time, so the installer does not turn speculation on.

From 32,768 tokens of context, a guard measures the speed. When speculation is slower than single steps, the guard stops it. To turn speculation off, set `KINDI_SPEC=0`.

CHAPTER 7

A more capable model

The 27B is the most capable Bonsai model. For harder work, Kindi 0.4.0 runs a larger model: AH-Nanbeige4.3-33B IQ3_XXS. This is our IQ3_XXS quantization of Nanbeige4.3-33B, made with bartowski's imatrix. The base model has 33 billion parameters. The file is 13.1 GB. It is a separate download from Hugging Face:

https://huggingface.co/ahtx/AH-Nanbeige4.3-33B-IQ3_XXS-GGUF

The model thinks before it answers, and it makes tool calls in the OpenAI format, so pi and other agents can use it in the same way as the Bonsai models.

Where does it run?

- **One card with 16 GB or more,** for example an RTX 3090 (24 GB).
- **Two cards with about 17 GB in total,** for example two RTX 30 series cards or two Tesla P100 cards.
- **One Quadro K6000 (12 GB),** with part of the model in host memory (see below).
- **About 17 GB of card memory in total.** Kindi puts the layers of the model on the cards in order: the first half on the first card and the second half on the second card. Only a small amount of data goes between the cards, so a PCIe x1 slot is sufficient. We measured it on two RTX 3080 10 GB cards and on two Tesla P100 cards.
- **A context of 16,384 tokens** on the RTX 3080 cards.

To get the model, type `kindi use nanbeige-33b`, or install with `install.sh --model nanbeige`. Both check the sha256 of the file after the download.

`kindi verify` compares the output of Kindi with the output of llama.cpp for the same file. With this model, 8 of the 11 test prompts were the same token for token.

The other 3 continued in the same way up to a near tie, where llama.cpp itself had two tokens with almost the same score. No prompt failed.

Before the server takes its first request, it does a self-check. It runs one prompt two times with single steps and one time with a check pass, and the tokens must be the same. If the check fails, the server stops with an error, and the service starts it again, so a fault at start-up cannot give a wrong reply. The check is on by default (`KINDI_NB_SELFCHECK=1`).

How fast is it?

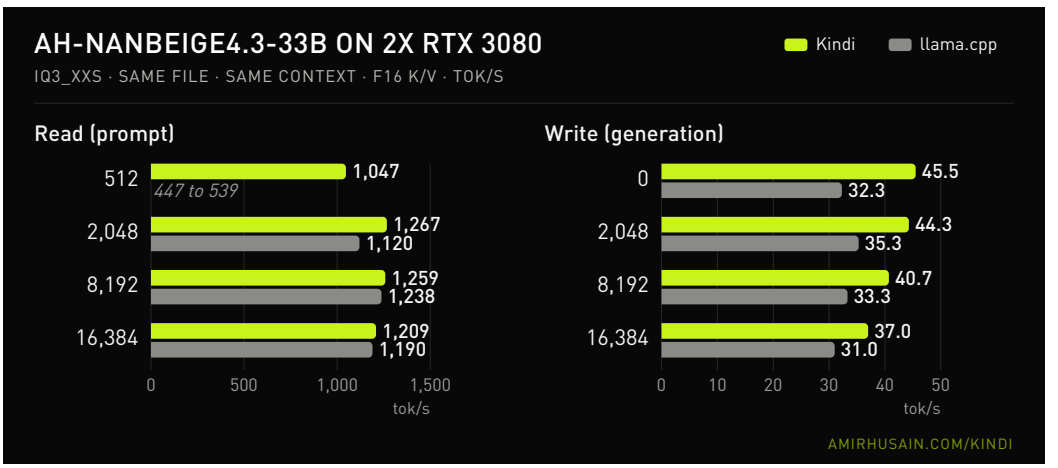


Figure 7-1. *Read and write speed of AH-Nanbeige4.3-33B IQ3_XXS on two RTX 3080 cards, with the same model file for both engines. Kindi is lime and llama.cpp is grey. With a 512-token prompt, the speed of llama.cpp changed from run to run.*

Write speed:

Context depth (tokens)	Kindi	llama.cpp	Kindi against llama.cpp
0	**45.5**	32.3	+41%
2,048	**44.3**	35.3	+25%
8,192	**40.7**	33.3	+22%
16,384	**37.0**	31.0	+19%

Read speed, a prompt of N tokens from an empty cache:

Prompt (tokens)	Kindi	llama.cpp	Kindi against llama.cpp
512	**1,047**	*447 to 539*	–
2,048	**1,267**	1,120	+13%
8,192	**1,259**	1,238	+2%
16,384	**1,209**	1,190	+2%

Conditions. Two GeForce RTX 3080 10 GB cards of our cluster, 190 W, PCIe x1, a host with 4 CPUs. Model AH-Nanbeige4.3-33B-IQ3_XXS.gguf (13.1 GB, sha256 9d4f18e8…) for both engines, 52 layers. f16 K/V cache for both. Kindi: 26 layers on each card, Kindi 0.4.0 development build (engine fde34d6, server 1ca5066), `kindi bench`, 64 tokens at each depth. llama.cpp: build 11382 (11fe02151), `llama-bench -ngl 99 -fa 1 -sm layer`, 128 tokens at each depth, 3 runs. `kindi verify` passed. 2026-10-30.

At a short context, Kindi writes 45.5 tok/s, and llama.cpp writes 32.3: Kindi is 41% faster. At 16,384 tokens, Kindi writes 37.0 tok/s, and llama.cpp 31.0. Kindi reads a 512-token prompt at 1,047 tok/s, and llama.cpp at 447 to 539. From 8,192 tokens, the two engines read at almost the same speed.

Through the server, with the OpenAI API, Kindi writes 45.4 tok/s greedy, and 44.2 tok/s sampled (temperature 0.7, top-k 20, top-p 0.8). The sampler finds the candidate tokens on the card, so only a few scores go over the PCIe x1 link.

On two Tesla P100 cards, Kindi writes the 33B at 21.0 tok/s at 512 tokens, about 2.2 times as fast as llama.cpp (Tesla P100 16 GB). The model also runs on a single RTX 3090 (24 GB).

It is a strong model for coding work, and it can do even better on a slightly faster GPU.

Can a Quadro K6000 run it?

Yes. The 33B file (13.1 GB) is larger than the 12 GB of the K6000. Kindi keeps 13 layers of the model in host memory, at a context of 8,448 tokens. While the card computes the layers on the card, it copies the host layers in, and the CPU computes a part of the work of each host layer. The result is 2.3 times the speed of llama.cpp at 8K tokens, with 10 layers on the CPU:

Prompt (tokens)	Kindi	llama.cpp	Times as fast
480	**3.85**	2.19	1.76×
2,016	**3.55**	1.96	1.81×
8,160	**3.56**	1.52	2.34×

> **Conditions.** One Quadro K6000 12 GB (a virtual machine with PCI pass-through), driver 470.256.02, a context of 8,448 tokens, an f16 K/V cache for both engines. Kindi: 13 host layers, the CPU computes 55% of the FFN rows of each host layer (8 threads). llama.cpp ad1b88c with `-ngl 42`. Write is the speed of 32 tokens after the prompt. `kindi verify`: 9 of 11 prompts token-exact, 2 end at a llama.cpp tie, 0 fail.

At a context of 1,024 tokens, Kindi writes 4.4 to 5.0 tok/s (two runs), and llama.cpp writes 2.5 to 2.7 tok/s.

How does prompt lookup help?

An agent copies much text: the contents of a file, the output of a command, a tool call that repeats earlier text. Prompt lookup uses this. Kindi finds the last earlier place in the prompt and the reply where the last few tokens occur. Its guess is the text that came after that place. The 33B then checks the guess in one pass, in the same way as with the 2B drafter. A guess from the prompt costs no model work.

Kindi measures how many guesses the model keeps, and it sets the length of each guess to give the most tokens for each unit of time. When guesses do not help, as in new prose, Kindi makes no guess. This is on by default.

Request	Greedy	Sampled (temperature 0.7)
Copy a file with changes	2.03×	1.95×
A tool call	1.17×	1.18×
Thinking, then code	1.04×	1.85×
Prose	1.00×	1.00×

> **Conditions.** Two RTX 3080 cards of our cluster, 190 W, PCIe x1, context 8,192. `kindi serve`, 400 tokens out, prompt lookup on (up to 5 tokens for each pass) against off. Kindi 0.4.0 development build (10df53d). The greedy replies were the same with and without prompt lookup. 2026-10-30.

When the model copies a file with changes, it writes 2.03 times as fast: 91.6 tok/s against 45.1. A tool call is 1.17 times as fast. Prose does not change.

How does it compare with the 27B?

In an initial test, we gave the 33B and the 27B the same tasks, one run of each task. The first test was the 65 tasks of the Kindi benchmark, with pi. The second test was 50 programming problems from LiveCodeBench, a public set of contest problems with hidden tests.

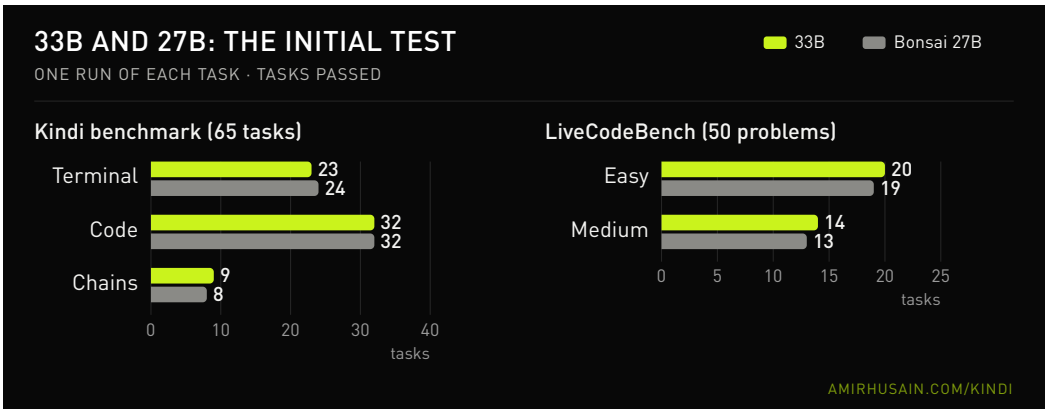


Figure 7-2. *Tasks that the 33B and the 27B passed in the initial test, one run of each task. The 33B is lime and the 27B is grey.*

Test	AH-Nanbeige4.3-33B	Bonsai 27B
Kindi benchmark: terminal	**23 / 24**	24 / 24
Kindi benchmark: programming and repair	**32 / 32**	32 / 32
Kindi benchmark: chains	**9 / 9**	8 / 9
Kindi benchmark: all	**64 / 65**	64 / 65
LiveCodeBench: easy	**20 / 21**	19 / 21
LiveCodeBench: medium	**14 / 29**	13 / 29
LiveCodeBench: all	**34 / 50**	32 / 50
Both tests	**98 / 115**	96 / 115

> **Conditions.** The 33B: llama.cpp (the official CUDA build) on two RTX 3080 cards of our desktop computer (WSL2), with our IQ3_XXS file. The 27B: Kindi 0.3.0 with its tensor-core settings and the 2B drafter, on two servers on RTX 3080 cards of our cluster at 190 W. Kindi benchmark: kbench 0.1.1 with pi 1.3.0, thinking on, 4,096 output tokens, context 16,384 on the pi side (the manifest of each run), 1,200 s for each task, one run of each task. LiveCodeBench: the 50 most recent easy and medium problems of its code_generation_lite set, one answer for each problem, thinking on, 4,096 output tokens, temperature 0.7, top-p 0.8, top-k 20, with the problem's public and hidden tests. 2026-10-29.

The two models passed the same number of Kindi benchmark tasks: 64 of 65. The 33B failed one terminal task, and the 27B failed one chain. On the LiveCodeBench problems, the 33B solved 34 of 50, and the 27B solved 32.

Can you use both models together?

You can use both models together. An agent tool such as opencode can send the fast steps to the 27B and the harder coding steps to the 33B, because both run on Kindi's OpenAI-compatible server.

CHAPTER 8

Can two cards work as one?

The weights of the 27B need about 6 GB, and the context needs more memory. One K1200 has 4 GB. Kindi joins two low-profile K1200s into one 8 GB AI card. It does the same with Nemotron.

What does tensor parallel mean?

Each layer of the model is a set of large matrices. Kindi cuts each matrix in two and puts one half on each card. For each token, the two cards do their halves at the same time. Then they exchange a small amount of data and join the results. So both cards work all the time. This method is tensor parallelism.

llama.cpp uses a different method on two cards: a layer split. It puts some layers on the first card and the other layers on the second card. The cards then work one after the other, and one card waits while the other card works.

How fast is it?

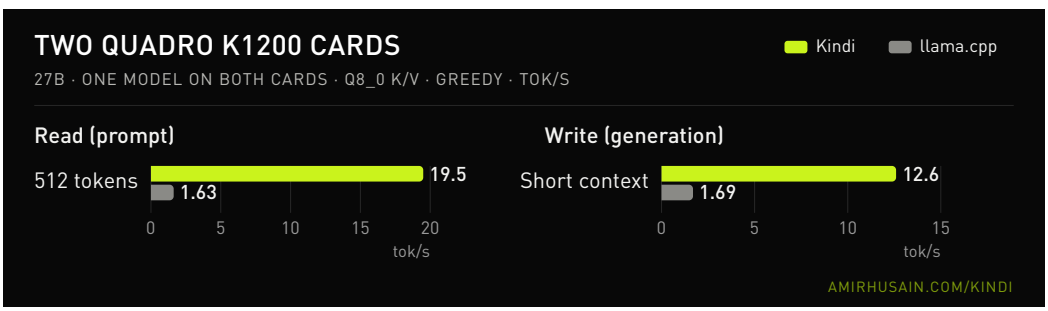


Figure 8-1. *Read speed with a 512-token prompt and write speed at a short context, for the 27B on two Quadro K1200 cards with the same model file. Kindi is lime and llama.cpp is grey.*

Measure	Kindi (tensor parallel)	llama.cpp (Vulkan, layer split)
Write, short context	**12.6**	1.69
Read, 512-token prompt	**19.5**	1.63
Context	24,576	*loads only with its default layer split*

> **Conditions.** Two Quadro K1200 4 GB cards in one computer. Model Ternary-Bonsai-2-27B-PTQ1_0.gguf (sha256 53107f53…) for both engines. q8_0 K/V cache for both, greedy. llama.cpp: Prism fork, Vulkan build (this computer has no CUDA toolkit for llama.cpp), flash attention on, its default layer split. Kindi: Kindi 0.2.0, tensor-parallel engine, exact head, 2026-10-23. Read: the same 512 token ids for both engines; llama.cpp llama-server with a context of 1,024 and no host prompt cache; Kindi chunked prompt path, best of 3 runs; 2026-10-22.

With the same model file on the same two cards, Kindi writes 12.6 tok/s and llama.cpp writes 1.69 tok/s. Kindi is 7.5 times faster. With the same 512-token prompt, Kindi reads 19.5 tok/s and llama.cpp reads 1.63 tok/s: Kindi is 12.0 times faster. llama.cpp loads the model only with its default layer split. With an even split, it runs out of memory. At 22,000 tokens, Kindi writes 9.84 tok/s on the two cards.

12.6 tok/s is faster than most people read, so you do not wait for the text.

Nemotron also runs split over the two cards:

Context (tokens)	Kindi read	llama.cpp read	Kindi write	llama.cpp write
512	**123**	15.3	**17.2**	7.44
8,000	**124**	19.0	**16.1**	7.37

> **Conditions.** The same two cards. Model NVIDIA-Nemotron-3.5-Bolt-E4B-UD-Q3_K_XL.gguf (sha256 3f8032b9…) for both engines. f16 K/V for both, the same method as the table above. Kindi: tensor parallel; the context fits 8,448 tokens; Kindi 0.3.0 development build (5350b87 with the server fix). llama.cpp: upstream Vulkan build 1b29621, its layer split. 2026-10-24.

Split over the two cards, Kindi writes Nemotron at 17.2 tok/s and llama.cpp at 7.44: Kindi is 2.3 times as fast. Kindi reads a 512-token prompt at 123 tok/s, and llama.cpp at 15.3. On two 4 GB cards, Nemotron's context is 8,448 tokens: the weights take 2.7 GB of each card's 4 GB.

Can each card run its own model?

Yes. Two cards can also be two separate servers, with one model on each card. Then two requests run at the same time, and each request runs at the speed of one card. The installer sets this up in one step: select one setup for each card, or a setup with `x2`, for example a 4B on each of two K1200 cards. Each server gets its own address.

Which is better? One model on both cards gives you a larger model for one user or one agent. One model on each card gives you two engines, for two agents or for a program that sends many short requests.

A worked example: Rainstorm



An HP EliteDesk 800 G4 SFF. Photo: eBay seller, used with permission.

OUR TEST COMPUTER

Rainstorm

Computer	A used HP EliteDesk 800 G4 small-form-factor (SFF) office computer
Cards	Two Quadro K1200 4 GB, low profile
Cost	**US$221.58** for all the parts
Power	The two cards take all their power from the slots. The computer needs no extra power cables.
Model	27B, tensor parallel, context 24,576
Speed	The 27B writes **12.6 tok/s** at a short context and 9.84 at 22,000 tokens

Rainstorm runs in text mode, with no desktop. We connect to it over the network, and pi or a script sends it requests.

Rainstorm shows what a used office computer can do. The installer offers it these setups:

What you want	The setup
The most capable model	the 27B split over both cards
A model from the second family	Nemotron split over both cards
Two jobs at the same time	one model on each card, for example a 4B on each card

We installed these three setups on rainstorm with the installer, from an empty folder. Each one passed `kindi verify` and a chat request, and Nemotron also answered a tool call.

CHAPTER 9

One small box for everything

The chapters before this one are about graphics cards. This chapter is about a computer with no graphics card at all: a mini PC with an AMD Ryzen 9 7940HS. The graphics unit of this chip, the Radeon 780M, uses the main memory of the computer. With 64 GB of memory, it holds the 27B with room to spare, and the CPU next to it can run a second model.

	Our test computer
Computer	Minisforum Venus UM790 Pro mini PC, 0.8 litres
Chip	AMD Ryzen 9 7940HS: 8 cores, 16 threads, Radeon 780M graphics (RDNA3)
Memory	2 × 32 GB DDR5-5600 SODIMM (64 GB, the largest that it takes)
Storage	Samsung 990 PRO 2 TB NVMe
Network	2.5 Gb Ethernet, Wi-Fi 6E, two USB4 ports
Power	one power adapter for the whole computer
Price	US$879 with 32 GB and 1 TB at the Minisforum store on 2026-10-24

Other makers sell mini PCs with the same Radeon 780M, from the Ryzen 7 7840HS, the Ryzen 9 7940HS and similar chips. Before you buy one for the 27B, check the memory: we measured the box with 64 GB.

What can one box do at the same time?

We ran three things on the box at the same time:

- the 27B on the Radeon 780M, with a context of 24,576 tokens;
- pi, the agent, which did tasks of the Kindi benchmark with the 27B;
- a second Kindi server with the 2B on the CPU, for short text jobs.

Measure	Alone	Together
27B on the 780M, read	88.0	75.3
27B on the 780M, write	10.5	9.21
2B on the CPU, read	62.6	15.5
2B on the CPU, write	42.5	7.8

Conditions. Kindi 0.3.0 development build (849840a). The 27B on the 780M with a context of 24,576; the 2B on the CPU with 8 threads. Each speed: a 2,000-token prompt and 256 tokens out, the median of 3 runs. "Together": the other engine has a request loop running. Then 10 tasks of the Kindi benchmark with pi on the 27B, while the CPU 2B was kept busy. 2026-10-25.

The 27B keeps most of its speed when the CPU 2B works at the same time: it writes 9.2 tok/s, against 10.5 when it runs alone. The 2B loses most of its speed: it writes 7.8 tok/s, against 42.5 alone. pi and the 27B passed all 10 tasks while the 2B was busy, with a median of 59 seconds for each task. The 2B answered 21 requests in that time.

The graphics unit and the CPU read from the same memory. A model writes each token by reading all of its weights from memory, so when both engines write at the same time, they share the memory speed of the box. Each one then writes a little slower than when it runs alone.

How does it compare with rainstorm?

Rainstorm (A worked example: Rainstorm) and this box are two ways to run the 27B for little money.

	Rainstorm	The UM790 Pro
What you buy	a used office computer and two used cards	a new mini PC
Cost	US$221.58 for all the parts	US$879 (32 GB) at the store; we tested 64 GB
The 27B writes	12.6 tok/s	10.6 tok/s at 512 tokens
Context of the 27B	24,576	24,576
At the same time	two models, one on each card	a second model on the CPU
Size	a small office computer	0.8 litres

Rainstorm costs less, and it writes the 27B a little faster. The mini PC is new, small and quiet, it runs from one adapter, and its memory leaves room for a second model and an agent next to the 27B.

What does the box need?

- **Linux and Docker.** The Kindi ROCm image holds ROCm, CuPy and Kindi. You do not install ROCm on the computer. The computer needs only the `amdgpu` driver, which is part of Linux.
- **One kernel option.** On the Radeon 780M, the driver can stop a running GPU program to let another one run, and on this chip that stop can fail and reset the graphics. In our tests, long prompts of 8,000 tokens and more made it happen, with Kindi and with llama.cpp. The kernel option `amdgpu.cwsr_enable=0` turns off this kind of stop, and with it the same tests passed three times out of three. The installer tells you what the option does and asks before it adds it. You then restart the computer one time.
- **Memory.** We measured the box with 64 GB, the most that it takes. The 27B and its context, the CPU 2B and pi ran together in it.

A brick of intelligence for the home

Put the box on a shelf next to the internet router. It is about 13 cm on each side and 5 cm high, smaller than a hardback book, and it costs about US$900. It connects with one network cable and one power adapter, and it stays on all day. It does not send your questions or your documents to a cloud service, so the work of the household stays in the house.

This one box holds all the parts of a home AI system: a large model for agent work, a small model for fast decisions, the agent that uses them, and the memory for all three. The other devices in the house do not need a model of their own. A laptop, a phone, a home-automation hub or a script on another computer sends its requests to the box on the local network, in the same OpenAI format that many programs already use. The box is the centre, and the devices around it are its hands and its screens.

A day with the box can look like this:

- **In the morning,** the 2B on the CPU sorts the mail and the messages of the night into short labels (Fast decisions on short text). Only the messages that need an answer go to the 27B, and pi writes a draft for each of them.
- **During the day,** a person asks many questions about one long document: a lease, an insurance policy or the manual of a new appliance. With the prompt cache, Kindi reads the document one time and then reads only each new question (What does the prompt cache do?).
- **In the evening,** a student works with pi and the 27B in a project folder. pi reads the files, runs the scripts and writes the report. The cache keeps the conversation of each person, so two people at two computers do not push each other out.
- **At night,** the 2B reads the text alerts of cameras, sensors and computers, and gives each one a single word: ignore, note or escalate. Only the escalations wake the 27B, which writes a short summary for the morning. Can one sentence make escalation safe? shows how to make the 2B lean the safe way.

One box also means less to look after. There is one computer to update, one to back up and one adapter to plug in. When the internet connection stops, the assistant continues to work, because it never needed the connection. When the household needs more, a second box or a computer with a card can join the network, and the installer sets up each one in the same way.

Can it travel?

The box is small, it has no moving parts other than its fans, and it runs on direct current from its adapter. So it can also go where the internet does not: in a camper van, on a boat or in a car. There, it gives the same assistant with no mobile signal. It can answer questions about the manual of the vehicle, keep a log of the trip and help a driver or a passenger with notes, plans and messages to send later.

We have not tested the box in a vehicle. If you try it, plan for these points:

- **Power.** A vehicle gives about 12 V. Use a DC-to-DC converter to the voltage of the box's adapter, or a small inverter for the adapter itself. Connect it so that the box stops when the engine stops, or it can drain the battery.
- **Heat and air.** Keep the vents open, and do not put the box in direct sun or in a closed compartment.

- **Movement.** Attach the box firmly, and use an NVMe drive (as in our box), not a disk with moving parts.

CHAPTER 10

What does the prompt cache do?

An agent sends the same text again and again. Each request starts with the system prompt, the list of tools and all the earlier turns, and only the end of the request is new. A server without a cache reads all of it again for each request. On a small card, that reading is most of the wait.

Kindi keeps the work that it did for one request and uses it again for the next request. When a request starts with the same tokens as an earlier one, Kindi reads only the new part. This is the prompt cache. It does not change what the model sees. With the cache, Kindi reads the new part of a prompt in other batch sizes, so the arithmetic can round a little differently. On a near tie between two tokens, a reply can then take the other token. In 295 turns of real agent conversations with Nemotron at temperature 0, 283 replies were the same with the cache on and off, and each of the other 12 took another reasonable path, for example `cat` in place of `read`.

When does the cache help?

Job	What stays the same	What Kindi reads again
A chat or an agent	the system prompt and the earlier turns	only the new message
Regenerate an answer	the full prompt	only the last token
Many questions about one document	the document	only the question
Two or more agents that take turns	the conversation of each agent	only the new message of each agent

How much faster is it?

We sent the same requests to one server two times: one time with the cache off and one time with it on.

		Nemotron, seconds			27B, seconds		
Job	Request	cache off	cache on	faster	cache off	cache on	faster
Agent, 2.2K-token system prompt	turn 2	2.76	**0.67**	4.1×	15.3	**2.52**	6.1×
	turn 3	2.96	**0.86**	3.4×	15.7	**2.93**	5.3×
Regenerate an answer	again	2.69	**0.43**	6.3×	15.5	**1.85**	8.4×
Questions about a 9K-token document	question 2	11.0	**0.7**	15.8×	49.4	**14.5**	3.4×
	question 3	10.7	**0.42**	25.5×	47.2	**11.5**	4.1×
Two agents that take turns	A, turn 2	2.58	**0.58**	4.4×	14.7	**2.61**	5.6×
	B, turn 2	2.57	**0.53**	4.8×	14.8	**2.79**	5.3×

> **Conditions.** GeForce GTX 1080 8 GB. Nemotron 3.5 Bolt E4B (UD-Q3_K_XL, sha256 3f8032b9…) and Ternary-Bonsai-2 27B (PTQ1_0, sha256 53107f53…). One server, one run with the cache off and one with it on. The times are for the full request: the prompt and a reply of up to 60 tokens (400 for the document questions, 600 for the regenerated answer). The replies were the same in all 12 requests of each model. Script: `examples/prompt_cache_bench.py`. Kindi 0.3.0 development build, 2026-10-24.

With Nemotron, the second turn of an agent comes back 4.1 times faster, a regenerated answer 6.3 times faster, and the third question about a 9,400-token document 25.5 times faster. The 27B gains as much or more on the agent turns: 6.1 times on the second turn. On the document questions, the 27B gains less, 3.4 and 4.1 times, because it continues from its last checkpoint at token 6,912 and reads the rest of the document again.

The first request of a conversation gains nothing. It is up to 8% slower with the cache on, because Kindi copies the cache to host memory while it reads the prompt.

Is it faster than the cache of llama.cpp?

llama.cpp also has a prompt cache. We ran the same requests against llama.cpp with its cache on, on the same card with the same model file. This table gives the

requests that each engine could take from its cache:

Job	Request	Kindi, s	llama.cpp, s	Kindi faster
Agent	turn 2	**0.67**	1.29	1.9×
	turn 3	**0.86**	1.3	1.5×
Regenerate an answer	again	**0.43**	4.93	11.5×
Questions about a document	question 2	**0.7**	3.16	4.5×
	question 3	**0.42**	3.1	7.4×
Two agents that take turns	A, turn 2	**0.58**	1.4	2.4×
	B, turn 2	**0.53**	1.44	2.7×

> **Conditions.** The same card, file and requests as the table above, Nemotron. llama.cpp: CUDA llama-server, build b11804 (the ggml-org server-cuda image of 2026-10-22), its prompt cache on. 2026-10-24. The first request of each job is not in the table: there, neither engine has anything in its cache.

On each cached request, Kindi answers faster. The difference is largest when the whole prompt is the same, as when you regenerate an answer: Kindi reads one token again and takes 0.43 s, and llama.cpp takes 4.93 s.

How does it work?

Kindi keeps three kinds of saved state.

1. **The conversation on the card.** The keys and values of every token stay on the GPU. A new request that starts with the same tokens continues from the end of the shared part.
2. **Checkpoints.** Some layers keep only the last part of the text (a sliding window), and the Bonsai models also keep a recurrent state. When a request goes back further than that, Kindi uses a checkpoint: a copy that it makes in host memory at regular steps of each prompt and at its end.
3. **Parked conversations.** When a new request shares less than half of the conversation on the card, that conversation moves to host memory. When its next turn comes, it moves back to the card. So agents that take turns each keep their own cache.

The Bonsai models can also save the start of a conversation to disk, so it is ready again after a restart. Each parked Nemotron conversation uses about 16 KB of host

memory for each token, plus about 42 MB for each checkpoint. The settings are in `docs/PROMPT-CACHE.md` in the Kindi repository.

CHAPTER 11

What does a power cap change?

A power cap is a limit on the power that a card can use. You set it with `nvidia-smi`. All ten RTX 3080 cards in our cluster run at a cap of 190 W, and their default limit is 380 W. We measured the two engines on one card at both caps.

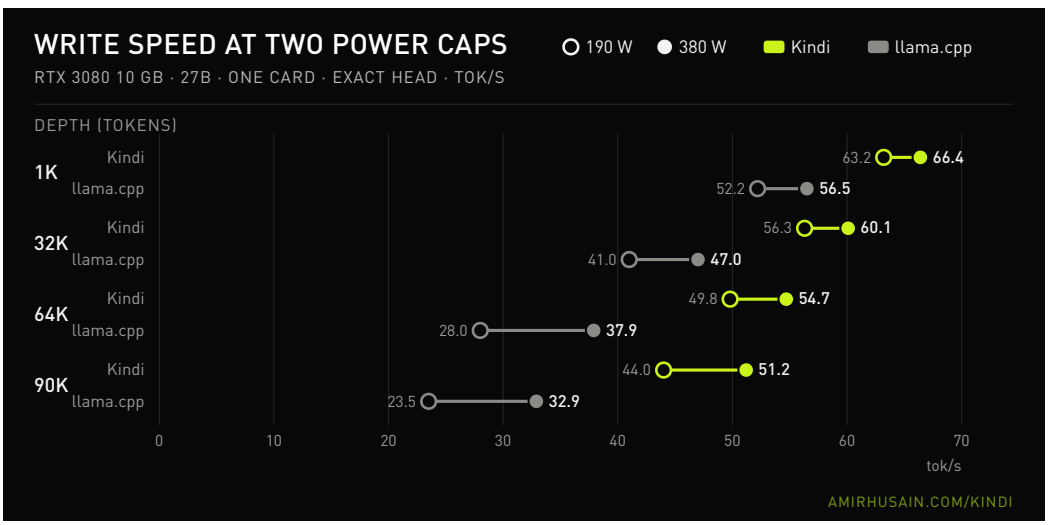


Figure 11-1. *Write speed of the 27B on one RTX 3080 at a 190 W cap (open dot) and a 380 W cap (filled dot), exact head. 190 W: Kindi 0.2.0; 380 W: dev build release-tcspec6 (2026-10-22). Kindi is lime and llama.cpp is grey.*

At half the power, Kindi writes 63.2 tok/s at a depth of 1K, against 66.4 at 380 W. At 90K, Kindi writes 44.0 against 51.2. llama.cpp writes 52.2 against 56.5 at 1K, and 23.5 against 32.9 at 90K. The read speed falls more with the cap. With a 512-token prompt, Kindi reads 1,093 tok/s at 190 W and 1,361 at 380 W.

What does that mean for you?

- **A cap costs little write speed.** To write one token, the card reads all the weights from its memory. So the write speed depends mostly on the speed of the memory, and less on the power of the arithmetic units.
- **A cap costs more read speed.** A long prompt is mostly arithmetic, and the arithmetic slows down at a lower power.
- **A cap lowers the heat and the load on the power supply.** At 380 W, our card used up to 375 W and reached 82 °C.
- **For agent work, a cap near 190 W is a good start.** With the 2B drafter on agent prompts, Kindi writes 99.9 tok/s at 190 W and 108 at 380 W.

To set a cap of 190 W on the first card, type:

```
$ sudo nvidia-smi -i 0 -pl 190
```

The cap stays until the next restart of the computer. To keep it, run the command again at each start, for example from a systemd service.

CHAPTER 12

How do you start on a new computer?

Kindi has a one-command installer. It finds your NVIDIA cards and your AMD graphics, shows each setup that your computer can run, with the best one first, and then downloads, checks and starts everything.

What do you need?

- A Linux computer (x86-64) with an NVIDIA card from this manual, AMD RDNA3 graphics such as the Radeon 780M, or no graphics: then the installer offers the CPU engine.
- For NVIDIA cards, the NVIDIA driver. Kepler cards such as the K420 need the 470 driver.
- For AMD graphics, only the `amdgpu` driver, which is part of Linux. ROCm is inside the Kindi image.
- Docker. If Docker or the NVIDIA Container Toolkit is missing, the installer asks you before it installs it. The installer also works with rootless Docker.
- Free disk space for the model: from 429 MB for the 2B to 5.9 GB for the 27B. Nemotron needs 4.2 GB, and the 33B needs 13.1 GB.

How do you install it?

First do a dry run. The dry run shows each step, each command and the list of setups, and it changes nothing:

```
$ curl -fsSL https://amirhusain.com/kindi/install.sh | bash -s -- --dry-run
```

Then install:

```
$ curl -fsSL https://amirhusain.com/kindi/install.sh | bash
```

To check the script before it runs, download it, compare its sha256 with the value in `docs/INSTALL.md`, read it, and then run it with `bash install.sh`.

How do you select a setup?

The installer shows a list. On a computer with an RTX 3060 and a GTX 1080, the list starts like this:

```
[3] Choose the setup
    1) every GPU, one server each: 27B on GPU 0 + 27B on GPU 1 (recommended)
    2) Ternary-Bonsai-2 27B on GPU 0 (NVIDIA GeForce RTX 3060)
    3) NVIDIA Nemotron 3.5 Bolt E4B on GPU 0 (NVIDIA GeForce RTX 3060)
    ...
    7) Ternary-Bonsai-2 27B on GPU 1 (NVIDIA GeForce GTX 1080)
    ...
    12) Ternary-Bonsai-2 2B on the CPU (80 threads; slow, no GPU needed)
    Choose [1]:
```

- Type one number for one server.
- Type two or more numbers, for example `2,8`, for two servers side by side: one model on each card.
- Type `5x2` to run setup 5 on two cards of the same kind, for example a 4B on each of two K1200 cards.

Each server gets its own container (`kindi`, `kindi-2`, …) and its own port, from 8093 up. A card can have only one server.

To install with no questions, add `--auto`: the installer takes the recommended setup and continues. To select a setup in a script, add `--config` and the numbers, for example `--config 2,8`.

The installer changes the kernel only when you answer at the keyboard. The gpu-unlock module for RTX 30 cards and the kernel option for the Radeon 780M both need your answer, also with `--yes` and `--auto`.

What does the installer do?

Step	What it does
1. Check the computer	Finds Linux, the memory, the free disk, the driver, and each NVIDIA and AMD graphics unit
2. Docker and GPU access	Finds Docker and the NVIDIA Container Toolkit or the AMD devices, and asks before it installs anything
3. Select the setup	Shows each setup for your computer, the best one first (above)
4. Build the image	Builds the Kindi image (one CUDA image for all NVIDIA cards, ROCm for AMD graphics) and checks that Kindi loads in it
5. Download the models	Downloads each file and checks its sha256
6. Convert and verify	Runs `kindi convert`, then `kindi verify` one time for each model on each kind of card
7. Start the servers	Starts each container on `127.0.0.1` and waits until it answers

The recommended setup for each kind of computer:

Your computer	Recommended	Context (tokens)
No graphics	2B on the CPU	24,576
Quadro K420 1 GB	2B	24,576
Quadro K420 2 GB	4B	23,808
One Quadro K1200	9B	fitted to the free memory
Two Quadro K1200 cards	27B, split over both cards	24,576
GeForce GTX 1080	27B	65,536
GeForce RTX 3080	27B, Tensor Cores and the 2B drafter	fitted to the free memory
Radeon 780M	27B	24,576
Two or more cards that do not split one model	the best model on each card, one server each	as above, for each card

To see only the setups with one model, add `--model` and `2b`, `4b`, `9b`, `27b`, `nemotron` or `nanbeige`. To select a context, add `--ctx` and a number. The servers listen on 127.0.0.1 only, and the installer asks for no keys.

At the end, the installer shows the address, the model and the context of each server, and the settings for pi. Then continue with Meet pi. Use the model `id` and the `contextWindow` that the installer shows in the pi settings of that chapter.

To remove Kindi, type `bash install.sh --uninstall`. This removes every server that the installer started. Your models stay in `~/.kindi`.

On NVIDIA cards of compute capability 8.0 and later, the installer turns on the 2B drafter for the 27B and downloads the MTP head of the 2B (How does a draft model make the 27B faster?). On these cards, with about 17 GB of card memory in total, the installer also offers the 33B (A more capable model).

What are the first steps after the install?

1. Find the port of the server. At the end, the installer shows the port of each server. The first server uses port 8093, the second server uses 8094, and so on.
2. Make sure that the server runs. Open `http://localhost:8093/health` in a browser, or type `curl -s localhost:8093/health`. The answer shows the model, the context size and the server state.
3. Connect a chat client. Kindi uses the OpenAI API at `http://localhost:8093/v1`. In Open WebUI, LM Studio, Jan or pi, add an "OpenAI-compatible" connection with this address. The API key can be any text, unless you set an `api_key`. The model id is in `/health`, for example `bonsai2-2b`.
4. Send a test request with curl:

```
$ curl -s localhost:8093/v1/chat/completions     -H "Content-Type:
application/json"     -d '{"model": "bonsai2-2b",
         "messages": [{"role": "user", "content": "Say hello."}]}'
```

1. Or send it with Python, after `pip install openai`:

```
from openai import OpenAI
client = OpenAI(base_url="http://localhost:8093/v1", api_key="none")
r = client.chat.completions.create(
    model="bonsai2-2b",
    messages=[{"role": "user", "content": "Say hello."}])
print(r.choices[0].message.content)
```

If the server is on a different computer, use its address in place of `localhost`. By default, the server accepts connections only from the same computer.

A request can also set `min_p`, `repeat_penalty` and `repeat_last_n`, in addition to `temperature`, `top_p` and `top_k`. These settings work as in llama.cpp, and they are off by default. `min_p` removes each token whose probability is less than that share of the most probable token. `repeat_penalty` makes the tokens of the last `repeat_last_n` tokens less probable, so the model repeats itself less.

CHAPTER 13

Meet pi

You type a task in plain language. Something reads your files, runs commands and writes the result. That something is pi, a coding agent for the terminal. pi sends your task to a language model, and the model answers with actions that use pi's four tools:

Tool	What it does
`read`	reads a file
`bash`	runs a shell command
`edit`	replaces exact text in a file
`write`	writes a whole file

pi runs each action, shows the result to the model, and lets the model choose the next action. The loop continues until the model says that the task is complete.

Because pi can use any server that speaks the OpenAI API, it can use Kindi. pi also has themes, and every picture in this manual uses the Kindi theme: lime on black.

THE PROMPT TYPED INTO PI

> contacts.txt is free text. Extract every e-mail address in it, remove duplicates, and write them to emails.txt sorted, one per line. Then print emails.txt.

THE FILE CONTACTS.TXT AT THE START

```
Meeting notes. Call ann@example.com or bob.smith@example.org about the budget.
CC: ann@example.com; zed+test@mail.example.net
No address on this line.
```

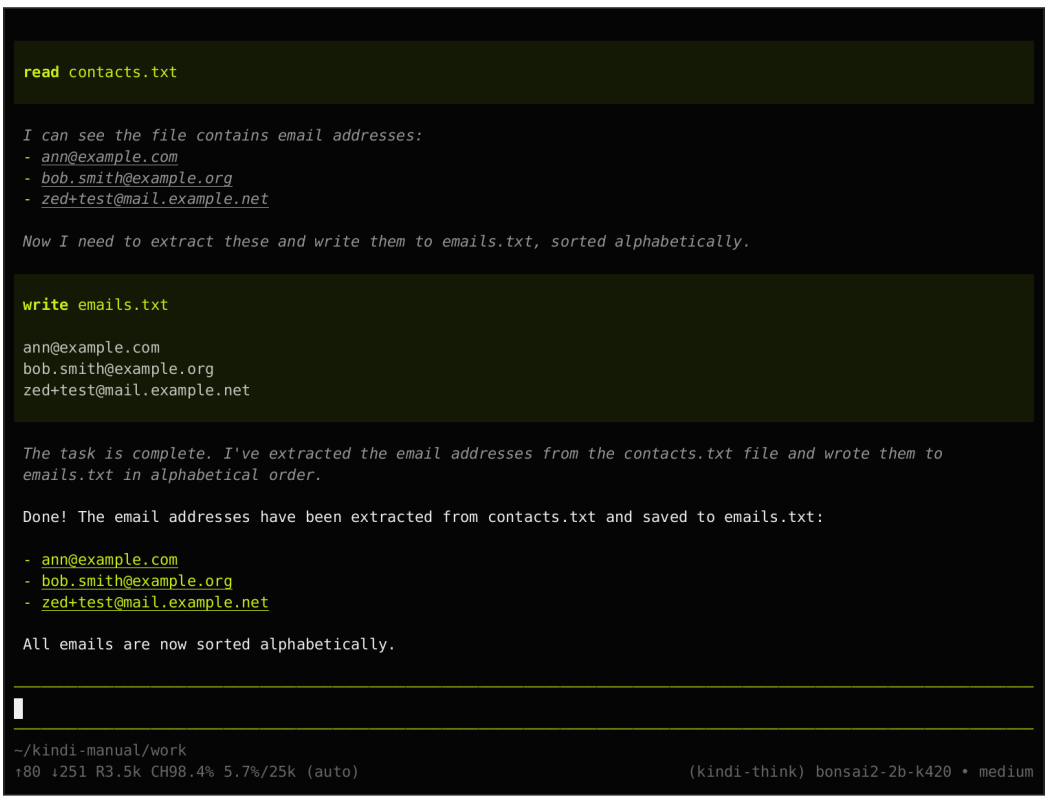


Figure 13-1. *pi on Kindi extracts e-mail addresses from free text (Quadro K420, live session)*

How do you connect pi to Kindi?

Add a provider for Kindi to `~/.pi/agent/models.json` (here, the 2B on the K420):

```
{
  "providers": {
    "kindi": {
      "baseUrl": "http://127.0.0.1:8093/v1",
      "api": "openai-completions",
      "apiKey": "none",
      "compat": {
        "supportsDeveloperRole": false,
        "supportsReasoningEffort": false,
        "thinkingFormat": "qwen-chat-template",
        "thinkingTokenBudgetField": "thinking_budget_tokens"
      },
      "models": [{
        "id": "bonsai2-2b", "name": "Bonsai 2 2B on Kindi", "reasoning": true,
        "input": ["text"], "contextWindow": 24576, "maxTokens": 8192,
        "cost": {"input": 0, "output": 0, "cacheRead": 0, "cacheWrite": 0}
      }]
    }
  }
}
```

The installer prints this block for each server that it starts, with the right address, model `id` and `contextWindow`. For Nemotron, the model `id` is `nemotron-bolt-e4b`.

Then give requests plenty of time in `~/.pi/agent/settings.json`. An agent's first prompt holds 1,000 to 5,000 tokens. On the K420, Kindi reads it once, at 33.3 tokens per second for 512 tokens, and keeps it in its cache (What does the prompt cache do?).

```
{
  "defaultProvider": "kindi",
  "defaultModel": "bonsai2-2b",
  "defaultThinkingLevel": "medium",
  "retry": { "provider": { "timeoutMs": 1800000, "maxRetries": 1 } },
  "thinkingBudgets": {
    "minimal": 256, "low": 512, "medium": 1024, "high": 2048
  }
}
```

Set `maxTokens` to 8192: with 2048, the model often stopped in the middle of a file. Set the thinking level to `medium` for terminal work: it raised the terminal success rate from 40% to 58%.

CHAPTER 14

What is an agent?

A chat assistant talks. An agent works.

Ask a chat assistant a question, and it gives you an answer. Give an agent a goal, and it does the work: it reads your files, runs commands, writes results and checks them. You say what you want, and the agent chooses the steps.

What is it like to work with an agent that runs a small model? Think of a new assistant on the first day. A new assistant is fast, keeps going all day and does clear, small jobs well. On long jobs with many rules, the assistant makes mistakes, and sometimes the assistant tells you that a job is complete when it is not. So you give the agent clear instructions, and you check its work.

Which rules make it work?

These rules come from 1,269 recorded sessions with the 2B, the smallest model:

1. **Give one small job at a time.** One clear goal, one output.
2. **Name the output file.** "Write the answer to answer.txt" works better than "tell me the answer". The model can make a mistake in its chat answer, but you can check a file.
3. **Say the exact format.** "One address per line, sorted" is better than "a list".
4. **Start a new session for each task.** In pi, type `/new`. Old text in the context confuses a small model.
5. **Check the result yourself.** The model can say "it works" when it does not. In 414 of 731 sessions where the result was wrong, the model's last message said that the result worked or was correct.
6. **Break big jobs into steps.** Can a big job be broken down? shows you how.

THE PROMPT TYPED INTO PI (LEFT PANE)

> config.json is a nested settings file. Write the value of database.port to answer.txt (only the value), then print answer.txt.

THE FILE CONFIG.JSON AT THE START

```
{"app": {"name": "demo", "port": 8000}, "database": {"host": "db", "port": 5432, "user": "app"}}
```

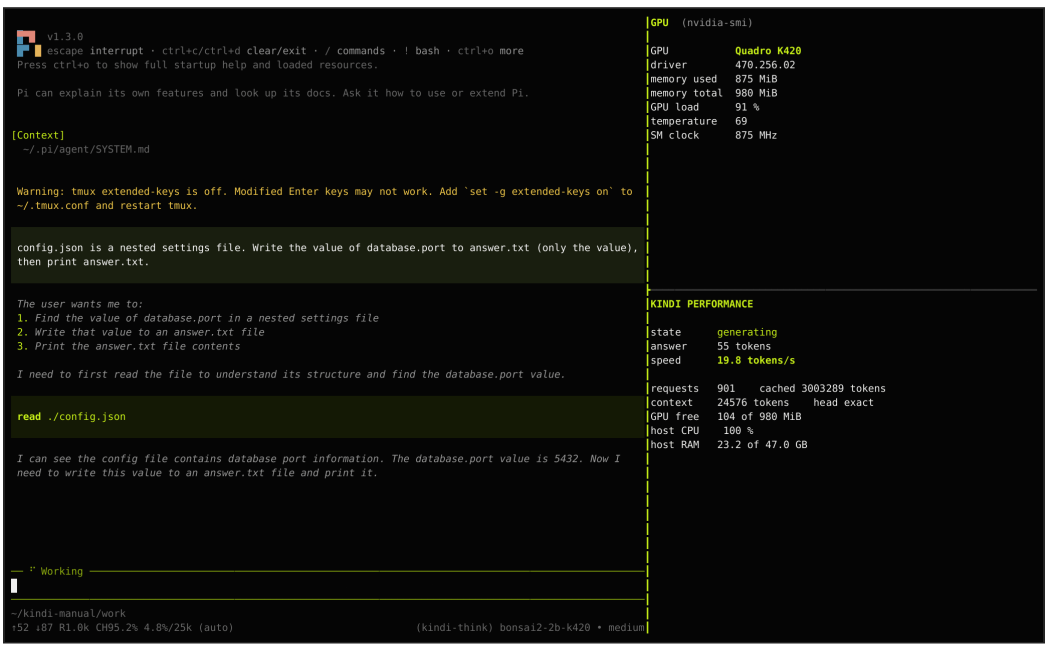


Figure 14-1. *A tmux setup: pi with Kindi on the left, the GPU at the top right, Kindi's live performance at the bottom right*

CHAPTER 15

Fast decisions on short text

Many useful jobs are simpler than agent work. A program sends a short text to Kindi and gets back one word or one corrected sentence: one request, one answer. Is this review positive? Is this e-mail spam? Must a person see this message now? Jobs like these suit a small model very well.

How well? We wrote 158 test items to find out: reviews, customer messages, e-mails, news sentences, sentences with spelling and grammar errors, and short texts to turn into tables. We ran each item once on the K420 at temperature 0, so the same input always gives the same answer.

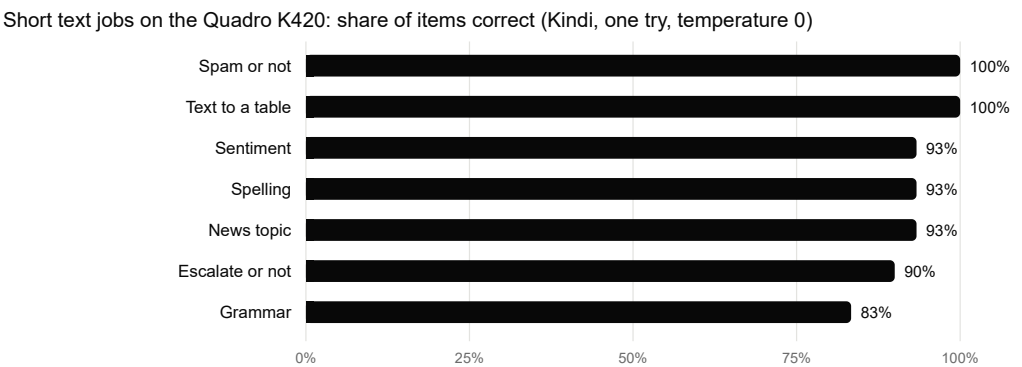


Figure 15-1. *Short text jobs on the Quadro K420: the share of items that Kindi got right*

Job	Correct (Kindi on the K420)	Time per item
Spam or not	**30/30**	2.4 s
Text to a table	**5/5**	4.2 s
Sentiment of a review	**28/30**	2.1 s
Spelling fixes	**14/15**	1.8 s
Escalate to a person or not	**27/30**	3.3 s
Topic of a news item	**28/30**	1.9 s
Grammar fixes	**10/12**	1.9 s

What do the answers look like?

The examples below are real answers from Kindi on the K420.

Spam or not. The program asks for one word: spam or ham.

INPUT
Hi Sam, attached are the slides for Thursday's meeting. Let me know if anything is missing.

KINDI
```
Ham
```

INPUT
CONGRATULATIONS!!! You WON a $1000 gift card. Click here to claim now!!!

KINDI
```
spam
```

INPUT
Your package from the post office is waiting. Pay the $1.99 fee at bit.ly/xx2 to release it.

KINDI
```
spam
```

Escalate to a person, or not. Use this to send urgent customer messages to a person at once.

INPUT
I can't log in after the update, can you help?

KINDI
```
No
```

INPUT
Your heater started smoking and there is a burning smell in my kitchen.

KINDI
```
Yes
```

INPUT
How do I change the email address on my account?

KINDI
```
No
```

Text to a table.

INPUT

Ann is 34 and lives in Austin. Bob is 29 and lives in Boston. Cy is 41 and lives in Chicago.

KINDI

```
| Name | Age | City |
|------|-----|------|
| Ann | 34 | Austin |
| Bob | 29 | Boston |
| Cy | 41 | Chicago |
```

INPUT

Apples cost $2, pears cost $3 and plums cost $4 per kilo.

KINDI

```
| Fruit | Price per kilo |
| :--- | :--- |
| Apples | $2 |
| Pears | $3 |
| Plums | $4 |
```

Headline and summary for a news story.

INPUT

The city council voted 7 to 2 on Monday to build a new bus line that links the airport with the central station. Work starts in spring and the line should open in two years. Supporters say it will cut traffic; opponents worry about the cost of 120 million dollars.

KINDI

```
Headline: City Council Votes for New Bus Line
Summary: The city council approved a new bus line linking the airport to the central station, with work starting in spring.
```

INPUT

A local bakery that opened in a garage five years ago now employs 40 people. The owners credit their success to sourdough bread and a delivery service by bicycle. They plan to open a second shop next month.

KINDI

```
Headline: Local Bakery Grows to 40 Employees
Summary: A garage-turned-bakery now employs 40 people and plans to open a second shop.
```

Clean up notes.

INPUT

meeting w/ supplier tues - prices up 5% from jan?? need to check contract. also ask about delivery delays (3 wks last time). Sara to send numbers by fri. budget review next week, invite finance

KINDI

```
- Meeting with supplier on Tuesday: review price increase of 5% from January and check contract.
- Ask supplier about delivery delays (last time took 3 weeks).
- Sara to send contact numbers by Friday.
- Budget review scheduled next week with finance.
- Invite finance to budget review.
```

Can one sentence make escalation safe?

In the escalation test, our first prompt missed 3 of the 15 messages that a person had to see at once. One of them said that a father had already taken pills with the wrong label. A missed escalation is the most dangerous error in this job. So we added one sentence to the prompt: "When you are not sure, answer yes."

Prompt (Kindi on the K420)	Correct	**Missed escalations**	False alarms
The rules only	27 of 30	**3**	0
The rules, plus "When you are not sure, answer yes."	30 of 30	**0**	0

That one sentence removed every missed escalation, and in this test it added no false alarms. With other messages, a sentence like this can add some false alarms: routine messages that a person must then look at. A few false alarms are the right trade. A false alarm costs a person a few seconds; a missed escalation can cost much more. (We also tried a long list of trigger words. The list was much worse: it missed 11 of the 15 urgent messages.)

The lesson works for any job where one type of error is worse than the other: tell the model which way to lean when it is not sure. And let your program check that each answer is one of the words that you allow, so that any other answer counts as "not sure".

CHAPTER 16

What works, and what to avoid?

This chapter gives the detailed results of the 2B, the smallest model. For each type of activity, the chapter tells you what the 2B does reliably, what it does only sometimes, and what you should not give it at all. Which model for which job? compares the 2B with the larger models. Each number is the share of sessions where the result passed an automatic check: "5/5" means five tries and five correct results.

Each table shows two setups: thinking off (with an output limit of 8,192 tokens) and thinking on (medium, with an output limit of 4,096 tokens). A task earns "Works" when one of the two setups passed at least 80 percent of the time, "Sometimes" at 30 to 79 percent, and "Avoid" below 30 percent. The appendix gives the results for every setup that we tested.

These rates come from many runs on our fast test machine (see How did we test?). The last column is one run of the same task with Kindi on the Quadro K420 itself, with thinking on for the terminal tasks and thinking off for the programming tasks. One run cannot give you a rate, but it does tell you whether the task also worked on the card. A dash means that the task has no K420 result.

Each table starts with the tasks that work. Each row that says Sometimes or Avoid also tells you what to do instead. For a terminal task, this is a command for your program to run, and that command passes the same hidden test. For a programming task, it is a program that runs the tests and tries again, or a library that already does the job.

Can it handle files and folders?

Find, list, rename, pack and unpack files. Overall: **43%** of 70 sessions passed on the fast machine.

Result	Task and the prompt that we gave	Thinking off	Thinking on	K420, one run
✓ WORKS	**Rename every .jpeg file in a folder to .jpg** *In the photos folder, rename every file that ends in .jpeg so that it ends in .jpg instead.*	2/5	5/5	✕ fail
✓ WORKS	**Make a .tar.gz backup of a folder** *Make a gzip-compressed tar archive named backup.tar.gz that contains the docs folder.*	3/5	5/5	✓ pass
✓ WORKS	**Make a script executable, run it, save the output** *Make run.sh executable, run it, and save everything it prints into out.txt.*	5/5	5/5	✓ pass
~ SOMETIMES	**Unzip an archive and count its files** *Extract archive.zip into a folder named out, then write the number of files to answer.txt.* **Do this instead:** Let your program run this command. It passes the same hidden test. `python3 -c "import zipfile; z = zipfile.ZipFile('archive.zip'); z.extractall('out'); print(sum(1 for n in z.namelist() if not n.endswith('/')))" > answer.txt`	1/5	2/5	✓ pass
✕ AVOID	**List the files that contain TODO** *List every file under the src folder that contains the word TODO, sorted, one per line.* **Do this instead:** Let your program run this command. It passes the same hidden test. `grep -rl 'TODO' src | sort > answer.txt`	1/5	0/5	✕ fail
✕ AVOID	**Count the lines of all .py files** *How many lines are there in total in all the .py files under src?* **Do this instead:** Let your program run this command. It passes the same hidden test. `find src -name '*.py' -exec cat {} + | wc -l > answer.txt`	0/5	1/5	✕ fail
✕ AVOID	**Find the 3 largest files** *Find the 3 largest files under the data folder, largest first, with their sizes.* **Do this instead:** Let your program run this command. It passes the same hidden test. `find data -type f -printf '%s %p\n' | sort -rn | head -3 > answer.txt`	0/5	0/5	✕ fail

Can it read logs and text?

Read logs, clean lists, find things in text. Overall: **42%** of 60 sessions passed on the fast machine.

Result	Task and the prompt that we gave	Thinking off	Thinking on	K420, one run
✓ WORKS	**Remove duplicate lines and sort** *names.txt has one name per line with repeats. Write unique.txt with each name once, sorted.*	5/5	5/5	✓ pass
✓ WORKS	**Extract e-mail addresses from free text** *Extract every e-mail address in contacts.txt, remove duplicates, write them to emails.txt sorted.*	4/5	5/5	✓ pass
✓ WORKS	**Count the words in a document** *How many words are in essay.txt (as wc -w counts them)?*	1/5	5/5	✓ pass
✕ AVOID	**Replace a word in many files** *In every .txt file under the notes folder, replace the word foo with bar.* **Do this instead:** Let your program run this command. It passes the same hidden test. `find notes -name '*.txt' -exec sed -i 's/foo/bar/g' {} +`	0/5	0/5	✕ fail
✕ AVOID	**Count the ERROR lines in a log** *logs/app.log is an application log. How many lines in it have the level ERROR?* **Do this instead:** Let your program run this command. It passes the same hidden test. `awk '$3=="ERROR"' logs/app.log | wc -l > answer.txt`	0/5	0/5	✕ fail
✕ AVOID	**Find the busiest client IP in a web log** *access.log is a web server log. Which client IP address made the most requests?* **Do this instead:** Let your program run this command. It passes the same hidden test. `awk '{print $1}' access.log | sort | uniq -c | sort -rn | head -1 | awk '{print $2}' > answer.txt`	0/5	0/5	✕ fail

Can it work with data files?

Read and change JSON, CSV and settings files. Overall: **80%** of 40 sessions passed on the fast machine.

Result	Task and the prompt that we gave	Thinking off	Thinking on	K420, one run
✓ WORKS	**Read one value from a nested JSON file** *config.json is a nested settings file. Write the value of database.port to answer.txt.*	5/5	5/5	✓ pass
✓ WORKS	**Change one line in a settings file** *settings.ini has a line port=8080. Change it to port=9090 and leave every other line as it is.*	5/5	5/5	✓ pass
✓ WORKS	**Convert CSV to JSON** *Convert people.csv to people.json: a JSON list with one object per row.*	5/5	4/5	✓ pass
~ SOMETIMES	**Sort a CSV file by a column** *scores.csv has a header row. Write sorted.csv with the rows sorted by score, highest first.* **Do this instead:** Let your program run this command. It passes the same hidden test. `(head -1 scores.csv; tail -n +2 scores.csv | sort -t, -k2,2nr) > sorted.csv`	0/5	3/5	✓ pass

THE PROMPT TYPED INTO PI

```
> config.json is a nested settings file. Write the value of database.port to
answer.txt (only the value), then print answer.txt.
```

THE FILE CONFIG.JSON AT THE START

```
{"app": {"name": "demo", "port": 8000}, "database": {"host": "db", "port": 5432,
"user": "app"}}
```

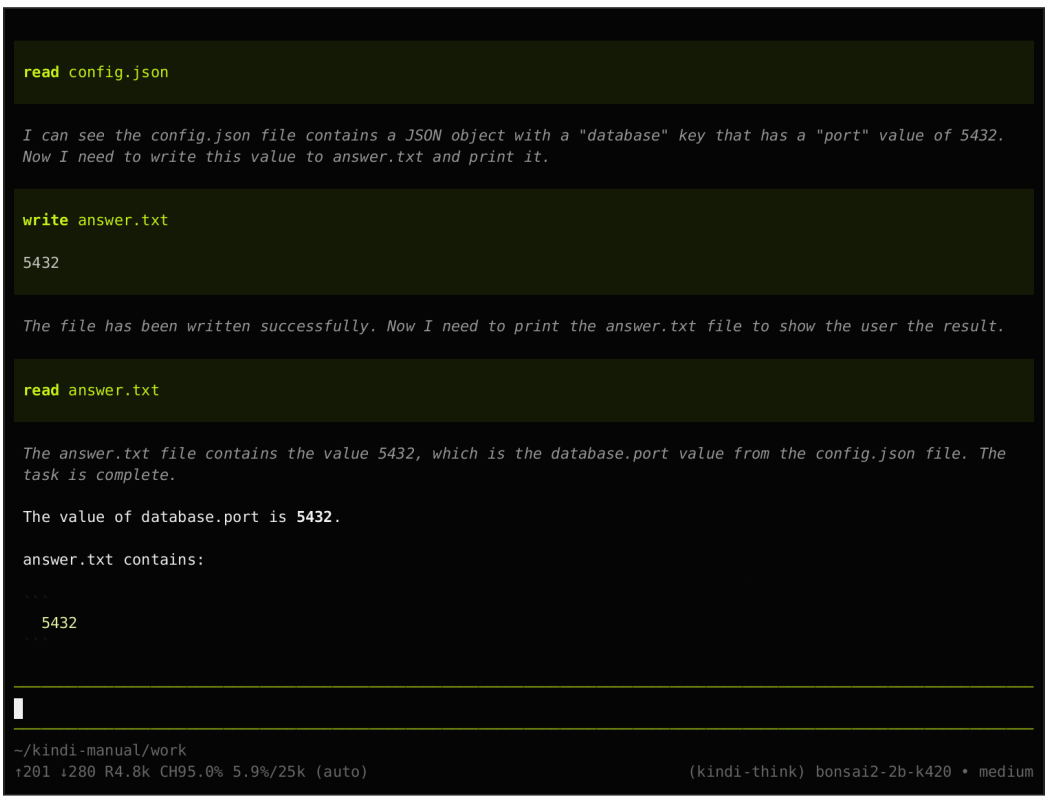


Figure 16-1. *pi on Kindi reads one value from a nested JSON file and writes it to answer.txt (Quadro K420, live session, thinking on)*

Can it look after the machine?

Ask about the machine, use git, plan jobs. Overall: **48%** of 60 sessions passed on the fast machine.

Result	Task and the prompt that we gave	Thinking off	Thinking on	K420, one run
✓ WORKS	**Make a folder a git repository and commit** *Make the project folder a git repository and commit README.md with the message: initial.*	5/5	5/5	✓ pass
✓ WORKS	**Report the installed Python version** *Which version of python3 is installed? Write only the version number to answer.txt.*	1/5	5/5	✓ pass
~ SOMETIMES	**Report the GPU name** *What is the name of the first NVIDIA GPU, as nvidia-smi reports it?* **Do this instead:** Let your program run this command. It passes the same hidden test. `nvidia-smi --query-gpu=name --format=csv,noheader -i 0 > answer.txt`	3/5	3/5	✓ pass
~ SOMETIMES	**Report the free disk space** *How much space is free on the filesystem that holds the working folder (df -k)?* **Do this instead:** Let your program run this command. It passes the same hidden test. `df -k --output=avail . | tail -1 | tr -d ' ' > answer.txt`	1/5	3/5	✗ fail
~ SOMETIMES	**Count the days between two dates** *How many days are there from 2026-01-15 to 2026-10-18?* **Do this instead:** Let your program run this command. It passes the same hidden test. `python3 -c "from datetime import date; print((date(2026, 10, 18) - date(2026, 1, 15)).days)" > answer.txt`	1/5	2/5	✗ fail
✗ AVOID	**Write a crontab line** *Write one crontab line to cron.txt that runs /opt/backup.sh every day at 02:30.* **Do this instead:** Let your program run this command. It passes the same hidden test. `echo '30 2 * * * /opt/backup.sh' > cron.txt`	0/5	0/5	✗ fail

THE PROMPT TYPED INTO PI

> Make the project folder a git repository and commit README.md in it with the commit message: initial. Then show git log --oneline for that folder.

THE FILE PROJECT/README.MD AT THE START

```
# Demo
```

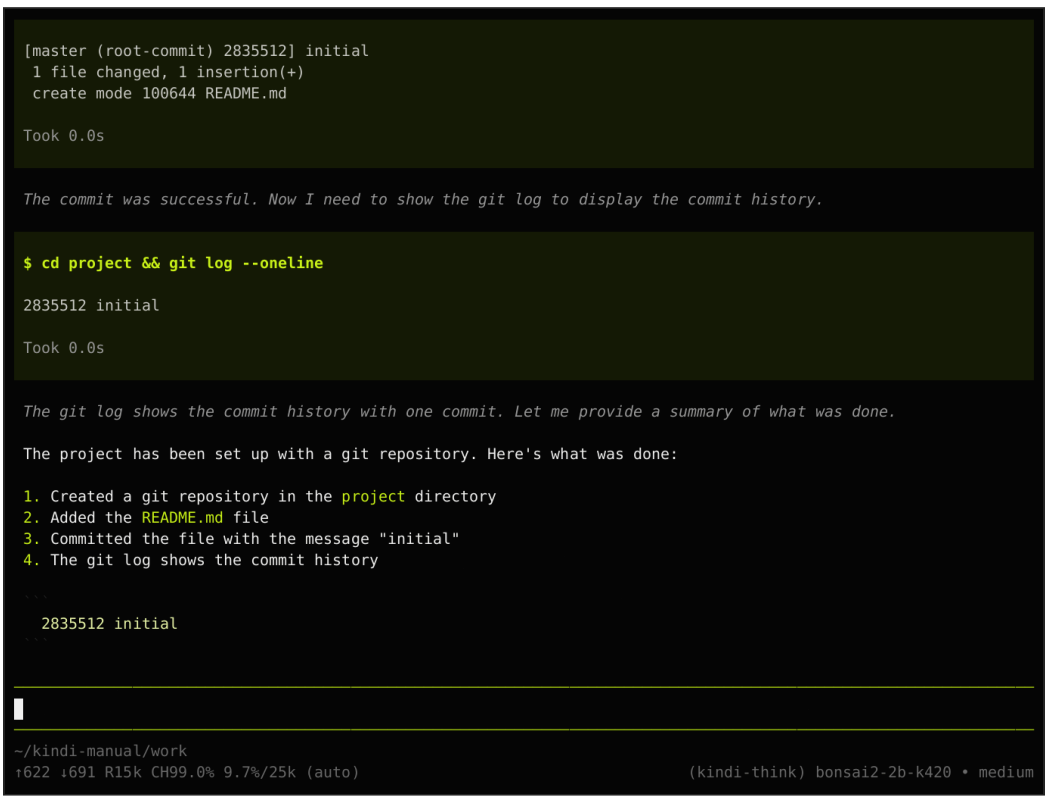


Figure 16-2. *pi on Kindi makes a folder a git repository, commits a file and shows the log (Quadro K420, live session, thinking on)*

Can it write shell scripts?

Write small bash scripts. Overall: **11%** of 28 sessions passed on the fast machine.

Result	Task and the prompt that we gave	Thinking off	Thinking on	K420, one run
~ SOMETIMES	**A script that greets a name from a variable** *Write hello.sh that prints Hello, NAME! where NAME comes from the environment, or world.* **Do this instead:** Let your program run this command. It passes the same hidden test. `printf '#!/bin/bash\necho "Hello, ${NAME:-world}!"\n' > hello.sh`	0/5	2/5	✕ fail
~ SOMETIMES	**A script that counts non-empty lines** *Create countlines.sh that prints the number of non-empty lines in a file, or an error.* **Do this instead:** Let your program run the tests and try again (Pattern 2 in Kindi in your own workflows). 5 of our 15 tries passed, and the program keeps the first version that passes.	0/3	1/3	–
✕ AVOID	**A script that adds its arguments** *Create sumargs.sh that prints the sum of its integer arguments, with an error for bad input.* **Do this instead:** Write it yourself: one loop over `"$@"` that checks that each argument is an integer. In our tests, 1 of 15 tries passed.	0/3	0/3	✕ fail
✕ AVOID	**A script that lists the 3 largest files** *Create largest.sh that prints the 3 largest files under a folder, with sizes.* **Do this instead:** Put the command from the files table in the script: `find DIR -type f -printf '%s %p\n' | sort -rn | head -3`. In our tests, 0 of 15 tries passed.	0/3	0/3	–

Can it write small Python programs?

Write one function from a clear description. Overall: **20%** of 84 sessions passed on the fast machine.

Result	Task and the prompt that we gave	Thinking off	Thinking on	K420, one run
✓ WORKS	**FizzBuzz as a list** *Create fizz.py with fizzbuzz(n) that returns 'Fizz', 'Buzz', 'FizzBuzz' or the number.*	3/3	2/3	✓ pass
✓ WORKS	**Limit a number to a range, with an error case** *Create clamp.py with clamp(x, lo, hi); raise ValueError if lo > hi.*	3/3	2/3	✓ pass
~ SOMETIMES	**Flatten nested lists** *Create flatten.py with flatten(x) for lists nested to any depth.* **Do this instead:** Let your program run the tests and try again (Pattern 2 in Kindi in your own workflows). 7 of our 15 tries passed, and the program keeps the first version that passes.	2/3	1/3	–
~ SOMETIMES	**Check balanced brackets** *Create brackets.py with balanced(s) for ()[]{}.* **Do this instead:** Let your program run the tests and try again (Pattern 2 in Kindi in your own workflows). 4 of our 15 tries passed, and the program keeps the first version that passes.	1/3	0/3	–
~ SOMETIMES	**A stack that also gives its minimum** *Create minstack.py with a MinStack class: push, pop, top, get_min.* **Do this instead:** Let your program run the tests and try again (Pattern 2 in Kindi in your own workflows). 3 of our 15 tries passed, and the program keeps the first version that passes.	0/3	1/3	–
~ SOMETIMES	**The most common words, with tie rules** *Create wordfreq.py with top_words(text, k), sorted by count, then alphabetically.* **Do this instead:** Let your program run the tests and try again (Pattern 2 in Kindi in your own workflows). 3 of our 15 tries passed, and the program keeps the first version that passes.	1/3	0/3	✗ fail

Result	Task and the prompt that we gave	Thinking off	Thinking on	K420, one run
~ SOMETIMES	**Group words that are anagrams** *Create anagram.py with group_anagrams(words), each group sorted.* **Do this instead:** Let your program run the tests and try again (Pattern 2 in Kindi in your own workflows). 2 of our 15 tries passed, and the program keeps the first version that passes.	1/3	0/3	–
✕ AVOID	**Binary search** *Create firstge.py with first_ge(arr, x) in O(log n), without bisect.* **Do this instead:** In your own code, use `bisect.bisect_left` from the Python standard library. It does this search. In our tests, 1 of 15 tries passed.	0/3	0/3	–
✕ AVOID	**Run-length encode and decode** *Create rle.py with encode('aaab') == 'a3b1' and decode.* **Do this instead:** Use `itertools.groupby` from the Python standard library. It groups the runs, and the encoding is then one line. In our tests, 0 of 15 tries passed.	0/3	0/3	–
✕ AVOID	**Merge overlapping intervals** *Create merge.py with merge(intervals).* **Do this instead:** Write it yourself: sort the intervals by start, then join each interval to the last one when they overlap. In our tests, 0 of 15 tries passed.	0/3	0/3	–
✕ AVOID	**A calculator for reverse Polish notation** *Create rpn.py with rpn('3 4 + 2 *') == 14 and errors for bad input.* **Do this instead:** Write it yourself: a list as the stack and one loop over the tokens. The `operator` module gives the operations. In our tests, 0 of 15 tries passed.	0/3	0/3	–

Result	Task and the prompt that we gave	Thinking off	Thinking on	K420, one run
✕ AVOID	**Read a matrix in spiral order** *Create spiral.py with spiral(matrix).* **Do this instead:** Write it yourself, and test it on an empty matrix, a 1x1 matrix and a 2x3 matrix. In our tests, 0 of 15 tries passed.	0/3	0/3	–
✕ AVOID	**Parse an INI file** *Create ini.py with parse_ini(text) that returns sections of keys and values.* **Do this instead:** Use the `configparser` module of the Python standard library. It reads INI files. In our tests, 0 of 15 tries passed.	0/3	0/3	–
✕ AVOID	**Find duplicate files by content** *Create dedupe.py with dedupe_files(folder).* **Do this instead:** Use `os.walk` to find the files, `hashlib` to hash each file, and a dict to group the paths with the same hash. In our tests, 0 of 15 tries passed.	0/3	0/3	–

Can it repair existing code?

Fix a known bug in a small file. Overall: **22%** of 18 sessions passed on the fast machine.

Result	Task and the prompt that we gave	Thinking off	Thinking on	K420, one run
~ SOMETIMES	**Fix a script that crashes on a CSV file** *report.py should print the total of the amount column of sales.csv, but it crashes. Fix it.* **Do this instead:** Let your program run the tests and try again (Pattern 2 in Kindi in your own workflows). 3 of our 15 tries passed, and the program keeps the first version that passes.	2/3	0/3	–

Result	Task and the prompt that we gave	Thinking off	Thinking on	K420, one run
~ SOMETIMES	**Fix two bugs in a statistics module** *stats.py: mean([]) crashes and median() is wrong for even lengths. Fix it.* **Do this instead:** Let your program run the tests and try again (Pattern 2 in Kindi in your own workflows). 7 of our 15 tries passed, and the program keeps the first version that passes.	1/3	1/3	✓ pass
✕ AVOID	**Fix bugs in a JavaScript word counter** *counter.js should count lowercase words. It has bugs. Fix them.* **Do this instead:** Fix it yourself. In our tests, the model did not find these bugs. In our tests, 0 of 15 tries passed.	0/3	0/3	–

Can it write JavaScript?

Node.js modules. Overall: **0%** of 18 sessions passed on the fast machine.

Result	Task and the prompt that we gave	Thinking off	Thinking on	K420, one run
✕ AVOID	**Make a URL slug from text** *Create slugify.js that exports slugify(s).* **Do this instead:** Use a tested package from npm, or write it yourself and test it on titles with symbols and extra spaces. In our tests, 0 of 15 tries passed.	0/3	0/3	–
✕ AVOID	**Merge two objects deeply** *Create deepmerge.js that exports deepMerge(a, b).* **Do this instead:** Use a tested package, for example the `merge` function of lodash. In our tests, 0 of 14 tries passed.	0/3	0/3	–
✕ AVOID	**Fix bugs in a word counter** *counter.js should count lowercase words. It has bugs. Fix them.* **Do this instead:** Fix it yourself. In our tests, the model did not find these bugs. In our tests, 0 of 15 tries passed.	0/3	0/3	–

Can it solve algorithms and parsers?

Classic programming problems. Overall: **3%** of 60 sessions passed on the fast machine.

Result	Task and the prompt that we gave	Thinking off	Thinking on	K420, one run
~ SOMETIMES	**Longest common subsequence** *Create lcs.py with lcs(a, b), dynamic programming.* **Do this instead:** Let your program run the tests and try again (Pattern 2 in Kindi in your own workflows). 8 of our 15 tries passed, and the program keeps the first version that passes.	0/3	2/3	–
✕ AVOID	**Shortest path in a graph** *Create dijkstra.py with shortest_path(graph, start, goal).* **Do this instead:** Let your program run the tests and try again (Pattern 2 in Kindi in your own workflows). 3 of our 15 tries passed, and the program keeps the first version that passes.	0/3	0/3	–
✕ AVOID	**Order tasks by their dependencies** *Create topo.py with order(n, deps), smallest order first, None for a cycle.* **Do this instead:** Use `graphlib.TopologicalSorter` from the Python standard library. It gives a valid order and reports a cycle. In our tests, 0 of 15 tries passed.	0/3	0/3	✕ fail
✕ AVOID	**Roman numerals in both directions** *Create roman.py with to_roman(n) and from_roman(s), strict.* **Do this instead:** Write it yourself from a table of values (1000 M, 900 CM, 500 D and so on), and test both directions. In our tests, 0 of 15 tries passed.	0/3	0/3	✕ fail

Result	Task and the prompt that we gave	Thinking off	Thinking on	K420, one run
✕ AVOID	**Evaluate arithmetic without eval** *Create calc.py with evaluate(s) for + - * / and parentheses.* **Do this instead:** Use the `ast` module to parse the expression, and allow only numbers, parentheses and the four operators. In our tests, 0 of 15 tries passed.	0/3	0/3	✕ fail
✕ AVOID	**Parse one CSV line by hand** *Create csvline.py with parse_line(line), with quoted fields.* **Do this instead:** Use the `csv` module of the Python standard library: `next(csv.reader([line]))` parses one line. In our tests, 0 of 15 tries passed.	0/3	0/3	✕ fail
✕ AVOID	**A least-recently-used cache** *Create lru.py with an LRUCache class.* **Do this instead:** Use `functools.lru_cache`, or `collections.OrderedDict` with `move_to_end`. In our tests, 0 of 15 tries passed.	0/3	0/3	✕ fail
✕ AVOID	**Split text into dictionary words** *Create wordbreak.py with sentences(s, words).* **Do this instead:** Write it yourself as a recursive function with `functools.cache`. In our tests, 0 of 15 tries passed.	0/3	0/3	✕ fail
✕ AVOID	**Multiply big numbers given as text** *Create bigmul.py with multiply(a, b), digit by digit.* **Do this instead:** In Python, `int(a) * int(b)` gives the exact product of numbers of any size. In our tests, 0 of 15 tries passed.	0/3	0/3	✕ fail
✕ AVOID	**The shortest window that contains given letters** *Create minwindow.py with min_window(s, t) in linear time.* **Do this instead:** Write it yourself with two pointers and a `collections.Counter`. In our tests, 0 of 15 tries passed.	0/3	0/3	✕ fail

What is the pattern?

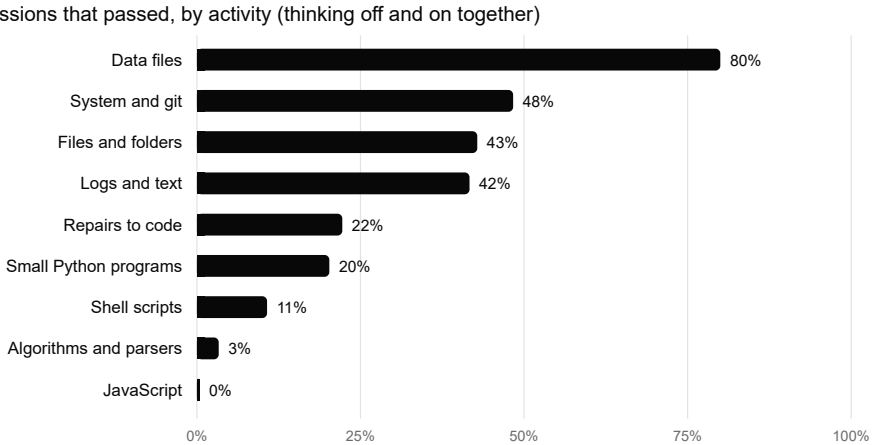


Figure 16-3. *Success rate by type of activity*

The pattern is clear. The 2B model does well when a task has **one action and one clear output**: read a value, remove duplicates, extract addresses, edit one line, make a commit. The model struggles when a task needs **several steps that depend on each other**: count, then sort, then select. JavaScript and real algorithms are also hard for the model.

CHAPTER 17

Can a big job be broken down?

Some jobs are too big for the 2B model as one task. You can still do them when you break them down the right way. We tested four ways to give the model a job that it failed every time as one prompt:

Task	One prompt	One prompt + "use commands"	Chain of prompts	Chain + one command per step
The busiest client IP in a web log	0/10	0/10	0/10	2/10
The number of ERROR lines in a log	0/10	0/10	0/10	2/10
The 3 largest files in a folder	0/10	0/10	0/10	0/10
All three	**0/30**	**0/30**	**0/30**	**4/30**

Passes / runs, thinking off and on together.

Better prompts alone did not save these jobs. A chain of small prompts did not help. When each step asked for one shell command, a few runs passed, but most did not. Why? The transcripts tell the story. The model often reads the data and then writes the result by hand, and on the way it changes a path or drops a line. When it counts or sorts, it makes small errors. A small model is good at language and at short decisions; exact work on data belongs to your program.

Which jobs does a breakdown rescue?

The chains above still asked the model to do exact work. What happens when your program does the exact work, and Kindi answers only one short question about each item? We tested five new jobs on the K420. Each job needs a judgment about

every item and then an exact count, list or sum. We wrote new items for this test, and we checked every result against the known answer:

Job	One prompt, API	One prompt, agent	Broken down, first prompt	Broken down, tested prompt
Tally *Count the positive, negative and neutral reviews among 20*	0/3	2/3	✓ **pass** *20 of 20 items right*	same prompt
Spam *Move the spam e-mails out of a folder of 20*	2/3	0/3	✕ fail *12 of 20 items right*	✕ fail *19 of 20 items right held-out set: fail, 9 of 10 items right*
Urgent *List the IDs of the 7 messages, among 20, that a person must see now*	0/3	2/3	✓ **pass** *20 of 20 items right*	same prompt
Topic *Find the news topic with the most headlines among 15, and its count*	0/3	0/3	✓ **pass** *13 of 15 items right*	same prompt
Orders *Add up the amount paid in 10 free-text order e-mails*	0/3	3/3	✕ fail *8 of 10 items right*	✓ **pass** *10 of 10 items right held-out set: pass, 6 of 6 items right*

One prompt with all the text failed almost every time: 2 of 15 runs passed. The model must judge every item and keep an exact count at the same time, and the count drifts by one or two.

The agent with one prompt passed 7 of 15 runs. The agent results were uneven. The agent got the order total right in all three runs and failed every spam and topic run. In one tally run, the agent counted correctly but wrote the three numbers without their labels, so the check failed.

Broken down, 3 of the 5 jobs passed with the first prompts that we wrote. The program asked Kindi one question for each item and did the counting itself. Each job took at most 73 seconds on the K420. In the topic job, Kindi gave 2 of the 15 headlines the wrong topic, but neither mistake changed the winning topic or its count.

The two failures teach the most important lesson of this chapter: **the question for one item must itself be tested.**

- **Order total.** The prompt ended with "for example 12.34", and the model copied the "12." into two answers: $42.00 came back as 12.00. Without the example number, Kindi read all 10 amounts correctly. Kindi also read all 6 amounts correctly in a held-out set that we wrote after the fix.
- **Spam sweep.** The prompt "Reply with one word: spam or ham." made the model call 8 normal e-mails spam. The tested prompt from Fast decisions on short text got 19 of the 20 e-mails right, and 9 of 10 in a held-out set. The job still failed, because the check needs every e-mail in the right folder.

With tested prompts, 4 of the 5 jobs passed. So before you run a per-item question on a thousand items, run it on ten items whose answers you know, and read the replies. Also decide what one wrong item costs: in a count of sentiment, one mistake moves a number by one; in a spam folder, one mistake hides a real e-mail.

What is the right way to split a job?

Split it into two kinds of steps:

- **Exact steps: your program does them.** Count, sort, filter, add, compare, select the top items. A few lines of Python or one shell command do this with no errors.
- **Language steps: Kindi does them.** Decide if a message is urgent. Find the sentiment. Write a short summary from facts that the program found.

Now the model only does the work that it does well. Fast decisions on short text shows how often each type of short decision is correct.

What does that look like in practice?

Here is a daily operations report, made on the K420 from an application log, a web log and eight customer messages. The program counts the errors and finds the

busiest clients. Kindi decides which messages are urgent, reads their sentiment and writes the summary.

```
levels = collections.Counter(l.split()[2] for l in open("app.log"))  ❶
clients = collections.Counter(l.split()[0] for l in open("access.log"))
top_ips = clients.most_common(3)
for msg in open("messages.txt"):
    urgent = "yes" in ask(RULES + "\n\n" + msg).lower()  ❷
    mood = ask("Classify the sentiment ... One word only.\n\n" + msg)
summary = ask("Write a summary for the operator in exactly three lines ... "
              "Use only these facts.\n\n" + facts)  ❸
```

❶ Exact: the program counts

❷ Language: Kindi decides

❸ Language: Kindi writes the words

And this is the report that the program wrote on the Quadro K420, with 17 calls to Kindi in 53 seconds:

```
# Daily operations report

## Summary (written by Kindi)

The ERROR lines are at 117 total, with the highest hour being 09 with 42
lines. The busiest clients are 10.0.0.2 with 153 connections. There are 3
urgent customer messages out of 8 total.

## Numbers (counted by the program)

- Log levels: {'ERROR': 117, 'INFO': 84, 'DEBUG': 101, 'WARN': 98}.
- ERROR lines by hour: {'08': 35, '09': 42, '10': 30, '11': 10}.
- Busiest clients: 10.0.0.2 (153), 10.0.0.5 (120), 10.0.0.7 (78).
- Urgent customer messages: 3 of 8.

## Customer messages (judged by Kindi)

| Urgent | Sentiment | Message |
|---|---|---|
| **YES** | negative | The checkout page has been down for all our customers
since 9 am. |
| **YES** | negative | Your app shows another customer's order history on my
account. |
| **YES** | negative | There is smoke coming out of the charger you sold me. |
| no | positive | Thanks, the new invoice layout is much clearer. |
| no | neutral | How do I export my data to CSV? |
| no | negative | The delivery was two days late and nobody told me. |
| no | positive | Can you add a dark mode? |
| no | positive | Great support today, thank you Maria! |
```

Look closely at how the work was divided, because it holds the most important lesson of this chapter:

- **The program counted every number.** The counts, the peaks and the busiest clients come from code, so they are exact.
- **Kindi flagged the three urgent messages and no others:** the outage, the privacy breach and the smoke.
- **Kindi wrote the summary from the facts that the program gave it,** and each number in the summary is a number that the program counted.

The model only judged messages and wrote words around facts. Keep that rule in your own workflows: let your program find and print every number, and let Kindi write only the words. Checking is then easy, because each number in the text must be one of the numbers in the facts. The full program is `workflow_demo.py` in the material for this manual.

How do you break down your own job?

1. **Write the job as a list of actions.** "Find the client with the most requests" is really three actions: get the client addresses, count them, pick the first.
2. **Mark each action as exact or language.** Counting, sorting and selecting are exact. Judging, classifying and writing are language.
3. **Write the exact actions as code.** You can ask Kindi to write a short function, and then test it on a small example before you trust it.
4. **Give Kindi one short decision or one short text at a time,** with the facts that it needs in the prompt.
5. **Check the result** with the rules in this chapter and in How do you get the best results?.

CHAPTER 18

Kindi in your own workflows

pi is one way to use Kindi. But what if you want full control? Because Kindi has an OpenAI-compatible API, any OpenAI client library works with it, and you can put the model inside a fixed program that you design.

Why does a fixed workflow help a small model?

Research on small models agrees with our tests. A small model does better in a fixed pipeline than as a free agent (Agentless, Xia and others, 2024). A program that checks each answer and tries again does better than a model that checks itself (Olausson and others, 2024). And many tries with an automatic check find a correct answer much more often than one try (Large Language Monkeys, Brown and others, 2024).

We saw the same thing in a quiz, where the model answered 12 questions about what a short Python program prints:

Method	Correct answers
One answer, thinking off	0 of 12
Majority of 5 answers, thinking off	3 of 12
One answer, thinking on	5 of 12
Majority of 5 answers, thinking on	5 of 12
At least one of the 5 answers correct	7 of 12

Look at the last line. The correct answer was often there, but a majority vote did not find it. If you can check an answer automatically, keep the first answer that passes the check.

Pattern 1: one call, one answer

Use this for short jobs: classify a line, extract a field, write a one-line summary.

```
from openai import OpenAI
kindi = OpenAI(base_url="http://127.0.0.1:8093/v1", api_key="none")

def ask(question, max_tokens=300):
    r = kindi.chat.completions.create(
        model="bonsai2-2b", max_tokens=max_tokens, temperature=0.2,
        messages=[{"role": "user", "content": question}])
    return r.choices[0].message.content.strip()

for line in open("tickets.txt"):
    label = ask("Classify this support ticket as BILLING, BUG or OTHER. "
                "Reply with one word.\n\n" + line)
    print(label, "|", line.strip())
```

Pattern 2: generate, check, try again

Use this for code and for any output that a program can check. The program runs the tests.

```
import subprocess

def generate_until_it_passes(task, tests, tries=5):
    for n in range(tries):
        code = ask(task + "\nReply with only the Python code.",
max_tokens=1500)
        code = code.strip("`").removeprefix("python").strip()
        open("solution.py", "w").write(code)
        r = subprocess.run(["python3", tests], capture_output=True,
                           text=True, timeout=60)
        if r.returncode == 0:
            return code  ❶
    return None  ❷
```

❶ The first version that passes the tests

❷ No version passed: ask a person

Should you feed the model its own error message, again and again? In our tests, a second and a third try with the test failures rarely fixed a program that was wrong.

Research agrees: a fresh try from the start often works as well as a repair (Olausson and others, 2024).

Pattern 3: the program does the steps, the model fills the gaps

Use this when a job has several steps. Your program does the steps that must be exact, and the model does only the part that needs language.

```
import collections, re
counts = collections.Counter(l.split()[0] for l in open("access.log"))  ❶
top_ip, hits = counts.most_common(1)[0]
errors = [l for l in open("app.log") if " ERROR " in l][:20]  ❷
summary = ask("Summarize these errors in three short lines for an operator:\n"
              + "".join(errors))  ❸
print(f"Top client: {top_ip} ({hits} requests)\n{summary}")
```

❶ Exact: the program counts

❷ Exact: the program filters

❸ Language: the model writes

How do you keep an agent safe?

An agent runs commands that a model wrote. What can go wrong? During our tests, programs that the model wrote did all of these things:

- One program used 26 GB of memory.
- One program printed text without end, and its output filled 22.8 GB of memory in the test program.
- One script started itself again and again: 42,592 copies of itself.

Our test machine stopped responding twice before we added limits. Use these limits for any agent that runs code:

1. **Run the agent in its own folder,** and never give it your home folder.
2. **Limit memory, file size and the number of processes** (`ulimit -v`, `ulimit -f`, `ulimit -u`, or a container).
3. **Write command output to a file with a size limit,** not into the memory of your program.
4. **Set a time limit** on each command and on each task.
5. **Stop every process of the agent** when the task ends.

CHAPTER 19

How do you get the best results?

- **Expect small tasks to take seconds.** On the K420, with thinking on, each terminal task of the 2B that passed took between 20 seconds and 2.5 minutes from start to end. On the GTX 1080, the 27B took 28 s for an average terminal task.
- **The first prompt is slow; the next ones are fast.** Kindi keeps the agent's prompt in its cache, so each later turn reads only the new part. The Bonsai models also save the start of the prompt on disk, so the second session starts in a fraction of a second. What does the prompt cache do? gives the numbers.
- **Give each agent its own conversation.** Kindi keeps a few conversations in host memory and moves each one back to the card when its turn comes, so two agents that take turns do not slow each other down much.
- **Use thinking for terminal work.** In our terminal tests of the 2B, thinking raised the success rate from 40% to 58%.
- **For code, try both.** In our programming tests of the 2B, thinking alone lowered the success rate from 15% to 10%.
- **Set the output limit to 8,192 tokens.** With 2,048, the 2B often stopped in the middle of a file.
- **Ask for the answer in a file.** In 32% of the 2B sessions without thinking, the model gave the answer in the chat and never wrote the file that the task asked for.
- **Use Python, shell and standard tools.** Avoid JavaScript with the 2B.
- **Keep files small on a small card.** The context is 24,576 tokens on the 1 GB K420, and a large file fills it fast. The GTX 1080 holds 65,536 tokens.

CHAPTER 20

Which model for which job?

Kindi 0.4.0 runs six models: the 2B, the 4B, the 9B and the 27B of the Ternary-Bonsai-2 family, Nemotron 3.5 Bolt E4B and AH-Nanbeige4.3-33B IQ3_XXS. We gave each model the same 65 agent tasks of the Kindi benchmark. The benchmark has 24 terminal tasks, 32 programming and repair tasks, and 9 chains of small prompts. This chapter compares the models and tells you when to use each one.

How do the models compare?

The table gives each model on a card that it fits. The last two rows give the 2B and the 4B on the same card.

Model	Card	Terminal	Programming and repair	Chains	Time per terminal task	Write speed, tok/s
2B	Quadro K420 1 GB	**13 / 24**	**7 / 32**	1 / 9	96 s	**20.8** (512 tokens)
4B	Quadro K420 2 GB	**22 / 24**	**23 / 32**	3 / 9	86 s	**9.8** (position 100)
9B	2× K1200, one server on each	**22 / 24 †**	**26 / 32**	–	37 s and 176 s	**23.4** (512 tokens, one K1200)
27B	GeForce GTX 1080	**24 / 24**	**32 / 32**	7 / 9	28 s	**29.4** (512 tokens)
Two models on the same card						
2B	Quadro K1200	**16 / 24**	**6 / 32**	0 / 9	25 s	**81.0** (512 tokens)
4B	Quadro K1200	**21 / 24**	**23 / 32**	6 / 9	24 s	**39.2** (512 tokens)

Conditions. kbench 0.1.x with pi, thinking on, 4,096 output tokens, one run of each task, 1,800 s for each task, exact head. The 2B ran on the Quadro K420 1 GB (context 24,576), and its terminal tasks ran on Kindi 0.1.0. The 4B ran on a Quadro K420 2 GB (context 25,088). The 9B ran with one 9B server on each of two K1200 cards (context 24,576). † Its terminal score is a composite. The first run used the fast head, and we ran the 6 failed tasks again with the exact head. The 27B ran on the GTX 1080 (Kindi context 65,536). Write speed, exact head: All rows Kindi 0.2.0, except the K420 2 GB (dev bb4008c).

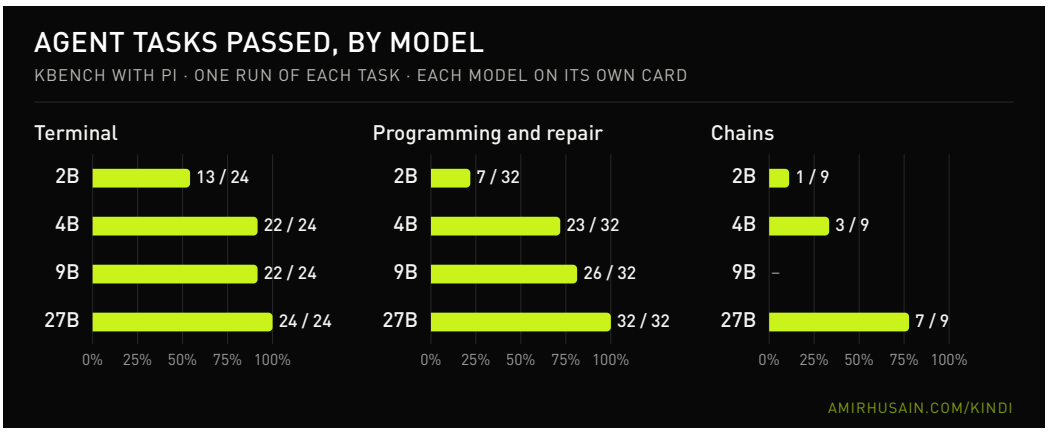


Figure 20-1. *The share of agent tasks that each model passed, by suite, each model on the card in the table above.*

Each step up the model ladder passes more tasks. The largest step is in programming: the 2B passed 7 of the 32 programming and repair tasks, and the 4B passed 23. The 27B passed 63 of the 65 tasks. On the same K1200 card, the 4B passed 23 programming tasks, and the 2B passed 6.

The 2B: short, clear jobs on a 1 GB card

What it does well. The 2B is fast on the smallest card: it writes 22.9 tok/s on the K420 (fast head). It is good at short decisions about a text. On the K420, it sorted spam from normal e-mail correctly in 30 of 30 cases and found the sentiment of a review in 28 of 30 cases. It also does terminal tasks with one action and one clear output: read a value, remove duplicates, extract addresses, edit one line, make a commit.

Where it struggles. Tasks with several steps that depend on each other, such as count, then sort, then select. It passed 1 of the 9 chains on the K420. JavaScript and real algorithms are also hard for the 2B.

When to use it. Use the 2B for short, well-defined tasks on a 1 GB card. Let your program do the exact steps, and let the 2B do the language steps. Can a big job be broken down? shows how. Chapters 14 to 18 give the full results of the 2B on the K420.

The 4B: agent work on a 2 GB card

What it does well. The 4B passed 22 of the 24 terminal tasks on the 2 GB K420, and 23 of the 32 programming and repair tasks. In all, it passed 48 of the 65 tasks.

Where it struggles. Chains: it passed 3 of the 9 on the 2 GB K420 and 6 of the 9 on a K1200. It is also slower: on the 2 GB K420, the 4B writes 9.8 tok/s, and the 2B writes 21.2 tok/s.

When to use it. Use the 4B for terminal work with several steps on a 2 GB card or on a K1200. Give it more time for each task than the 2B.

The 9B: the largest model on one small card

What it does well. The 9B passed 26 of the 32 programming and repair tasks and, as a composite, 22 of the 24 terminal tasks. On one K1200, it writes 23.4 tok/s at 512 tokens.

Where it struggles. It failed six programming tasks. Two were in JavaScript (a deep merge of nested objects and a URL slug). Four were exact algorithms in Python: multiplication of very large numbers, an arithmetic calculator, an INI parser and a task order with dependencies. The 9B needs a 4 GB card, and its largest context on one K1200 is 47,360 tokens.

When to use it. Use the 9B for programming tasks on one 4 GB card. With two K1200 cards, you can run one 9B server on each card, or the 27B on both.

The 27B: the most capable model

What it does well. On the GTX 1080, the 27B passed all 24 terminal tasks, all 32 programming and repair tasks, and 7 of the 9 chains. It took 28 s for an average terminal task. It writes 29.4 tok/s at 512 tokens on the GTX 1080. On an RTX 3080 at 190 W, it writes 63.2 tok/s at a depth of 1K, and 111.5 tok/s at 8K with the 2B drafter.

Where it struggles. The two chains that it failed both asked for the 3 largest files in a folder. The 27B needs 8 GB of card memory: a GTX 1080, two K1200 cards or an RTX 3080.

When to use it. Use the 27B for jobs with many steps and for code that must pass its tests. Also use it for long documents. The GTX 1080 holds 65,536 tokens of context, and the RTX 3080 about 85,000 (84,992). On a mini PC with Radeon 780M graphics, the 27B runs with a context of 24,576 tokens.

Nemotron: a second family

What it is. Nemotron 3.5 Bolt E4B is a model of a different design (Gemma 4) from NVIDIA. Its support in Kindi is experimental, and we are still optimizing it. Kindi reads the model's tool calls in its own format and turns them into the tool calls that pi and other OpenAI clients expect. On the RTX 3080, Kindi writes it at 123 tok/s, and on two K1200 cards at 17.2 tok/s, more than twice the speed of llama.cpp on the same cards.

When to use it. Use Nemotron when you want a second model family next to the Bonsai models. For the most reliable results, use the Bonsai models.

The 33B: the largest model

What it is. AH-Nanbeige4.3-33B IQ3_XXS is our quantization of Nanbeige4.3-33B. It needs NVIDIA cards of the RTX 30 series or later, with about 17 GB of card memory in total. On two RTX 3080 cards, Kindi writes it at 45.5 tok/s, 41% faster than llama.cpp with the same file. A more capable model gives the details.

When to use it. Use the 33B for harder agent work when you have two RTX 30 cards or later.

Which model should you use?

- **Use the 2B** for short, well-defined tasks on a 1 GB card or on the CPU. Let your program do the exact steps.
- **Use the 4B** for terminal work with several steps on a 2 GB card or a K1200.
- **Use the 9B** for programming tasks on one 4 GB card.

- **Use the 27B** for programming, long documents and jobs with many steps, on 8 GB of card memory or a Radeon 780M computer with enough memory.
- **Use Nemotron** (experimental) when you want a second model family.
- **Use the 33B** for harder agent work on two RTX 30 cards or later.

The installer shows the models that fit your hardware, and it lists the best one first. To see only one model, add `--model` (see How do you start on a new computer?).

What applies to every model?

- **Check the result yourself.** A model can say that a job is complete when it is not. With the 2B, in 414 of 731 sessions where the result was wrong, the last message of the model said that the result worked or was correct. Let a test or your program decide.
- **Give one job at a time, and name the output file.** These rules come from the 2B sessions in What is an agent?.
- **One Kindi server answers one request at a time.** Several agents on one card wait for each other, and the prompt cache keeps each agent's conversation. More cards give you more servers, and the installer starts one server on each card.
- **The 2B does not replace a large cloud model on hard problems.** Of our 10 hard programming tasks, the 2B solved only two, and only some of the time.
- **At 380 W on the RTX 3080, llama.cpp read a 22,000-token prompt 2% faster than Kindi** (dev build release-tcspec6). At 190 W on Kindi 0.2.0, the two engines read it at the same speed.

So you get a private assistant on a card that costs about US$15, a 27B model on a used office computer with two K1200 cards (US$221.58 in parts), and a 27B model, an agent and a second model on one small AMD computer.

CHAPTER 21

How did we test?

Which hardware and software?

- **The K420 machine.** An NVIDIA Quadro K420 (1 GB) with driver 470 runs Kindi 0.1.0 in its Docker image. All the screenshots, the K420 task results and the K420 timings in this manual come from this machine.
- **The fast test machine.** To run thousands of sessions in hours, we also ran the same model file (Ternary-Bonsai-2-2B, PTQ1_0) with llama.cpp on two RTX 3080 cards. In the short text jobs at temperature 0, the two engines gave the same answer for 156 of 158 items. We also ran the short text jobs, 37 tasks and every screenshot on the K420 itself, and the Kindi numbers in chapters 2 and 13 to 19 come from the K420.
- **The other cards.** Chapters 3 to 12 and 20 use Kindi 0.2.0, 0.3.0 and 0.4.0 builds on the 2 GB K420, the K1200, two K1200 cards, the GTX 1080, the RTX 3060 and the RTX 3080. Each table gives its card, model file, K/V cache, context, mode and build, and each comparison uses one card, one model file and one context.
- **The AMD computer.** A Minisforum UM790 Pro (Ryzen 9 7940HS, Radeon 780M, 64 GB) runs Kindi 0.3.0 in the Kindi ROCm image, and llama.cpp with its ROCm and Vulkan builds, with the same model files.
- **The CPU tests.** Rainstorm's Intel Core i7-8700 and the Ryzen 9 7940HS, with the same model files and the same number of threads for both engines.
- **The agent.** pi 1.3.0, with its four built-in tools, a short system prompt, a new session for each task, and the same sampling settings as Kindi (temperature 0.7, top-p 0.8, top-k 20).

Which tests?

Test	Tasks	Runs
Programming tasks (Python, shell, JavaScript, repairs, algorithms)	32	3 per task and setup, 5 setups
Everyday terminal tasks	24	5 per task and setup, 5 setups
Chains of small prompts	3	5 per task and setup, 3 setups
Short text jobs (through the API)	158 items	1 each, temperature 0
Tasks on the K420 itself (24 terminal, 13 programming)	37	1 each
Short text jobs on the K420 itself	158 items	1 each, temperature 0
The Kindi benchmark for each model (24 terminal, 32 programming and repair, 9 chains)	65	1 each
The prompt cache, cache off and on, and against llama.cpp (GTX 1080)	4 jobs, 12 requests	1 each
The Kindi benchmark for the 27B with the 2B drafter (RTX 3080)	65	1 each

Each task has a hidden test that the agent cannot see. Before we used a test, we proved it: it had to fail on the untouched folder and pass on a correct reference solution. A run passes only when the hidden test passes. What the agent says about its own work does not count.

How did we keep the results honest?

- **We threw away one full run.** In our first run, the agent could see the test files, and it often wrote its files to the wrong folder. We fixed both problems and ran everything again.
- **We did not count server failures as model failures.** When a model server stopped, the test program put the run back in the queue.
- **We limited what the agent's programs could do.** A watchdog stopped any program that used more than 2 GB of memory or started more than 200 processes, and the operating system limited the size of files. Without these limits, programs that the model wrote stopped our test machine twice.
- **We generated the numbers in this manual from the result files.**

You can run the same tests yourself. The tasks, the hidden tests, the reference solutions, the test runner and our result summaries are in the Kindi benchmark: https://github.com/ahtx/kindi-bench. The runner works with Kindi and with any other OpenAI-compatible server.

CHAPTER 22

Where can you read more?

- **Kindi.** Source code, install guide and benchmarks: https://github.com/ahtx/kindi. Project page: https://amirhusain.com/kindi.
- **The Kindi benchmark.** The tasks, hidden tests and runner behind this manual: https://github.com/ahtx/kindi-bench.
- **pi.** The coding agent: https://pi.dev.
- **Ternary-Bonsai-2.** The models, from Prism ML: https://huggingface.co/prism-ml/Ternary-Bonsai-2-2B-gguf, and the same address with 4B, 9B or 27B in place of 2B.
- **AH-Nanbeige4.3-33B IQ3_XXS.** The model card and the GGUF file: https://huggingface.co/ahtx/AH-Nanbeige4.3-33B-IQ3_XXS-GGUF.
- **The MTP head of the 2B.** https://huggingface.co/ahtx/kindi-mtp-heads. Its use in Kindi: `docs/MTP.md` in the Kindi repository.
- **Nemotron 3.5 Bolt E4B.** The GGUF file that Kindi uses, from unsloth: https://huggingface.co/unsloth/NVIDIA-Nemotron-3.5-Bolt-E4B-GGUF.
- **ROCm.** AMD's GPU software, inside the Kindi ROCm image: https://rocm.docs.amd.com.
- **gpu-unlock.** The third-party module for GeForce RTX 30 cards that the installer can offer: https://github.com/Zagitta/gpu-unlock.
- **Kindi on AMD graphics.** `docs/AMD.md` in the Kindi repository.
- **Kindi on each card.** The settings for each card: `docs/CARDS.md` in the Kindi repository. The installer: `docs/INSTALL.md`.
- **llama.cpp.** The reference engine that Kindi's output matches: https://github.com/ggml-org/llama.cpp.
- **vLLM.** The GPU requirement ("compute capability 7.5 or higher") is in its installation guide: https://docs.vllm.ai/en/latest/getting_started/installation/gpu/.

- **SWE-agent** (Yang and others, 2024). How the design of an agent's tools changes its success: https://arxiv.org/abs/2405.15793.
- **Agentless** (Xia and others, 2024). A fixed pipeline as an alternative to a free agent: https://arxiv.org/abs/2407.01489.
- **Large Language Monkeys** (Brown and others, 2024). More tries with a check find more correct answers: https://arxiv.org/abs/2407.21787.
- **Is Self-Repair a Silver Bullet for Code Generation?** (Olausson and others, 2024). Why new tries can beat repairs: https://arxiv.org/abs/2306.09896.
- **Large Language Models Cannot Self-Correct Reasoning Yet** (Huang and others, 2024): https://arxiv.org/abs/2310.01798.

Card photos and specifications

We use these images to identify each card. They are not photos of our own cards. Each line gives the image and the page that it comes from.

- Quadro K420: NVIDIA, the product image in the Quadro K420 data sheet, https://www.nvidia.com/content/dam/en-zz/Solutions/design-visualization/quadro-product-literature/13720_DS_NV_Quadro_K420_Aug25_US_NV_HR.pdf
- Quadro K1200 (two sheets): PNY, https://d2vfia6k6wrouk.cloudfront.net/productimages/171f62a8-250a-4f87-812b-b19b00eb84c7/images/pny-nvidia-quadro-k1200-dp-3qtr.png, from https://www.pny.com/nvidia-quadro-k1200-displayport
- GeForce GTX 1080: "NVIDIA GTX1080" by Benedikt S. Vogler, licensed under CC BY-SA 2.0 (https://creativecommons.org/licenses/by-sa/2.0/), cropped, https://commons.wikimedia.org/wiki/File:NVIDIA_GTX1080.jpg
- Quadro K6000: "NVIDIA Quadro K6000" by TopGear-V12, licensed under CC BY-SA 4.0 (https://creativecommons.org/licenses/by-sa/4.0/), resized, https://commons.wikimedia.org/wiki/File:NVIDIA_Quadro_K6000.jpg
- GeForce RTX 3080: NVIDIA, the product gallery of the RTX 3080 page, cropped, https://images.nvidia.com/aem-dam/Solutions/geforce/ampere/rtx-3080/geforce-rtx-3080-product-gallery-full-screen-3840-2.jpg, from https://www.nvidia.com/en-eu/geforce/graphics-cards/30-series/rtx-3080/

- HP EliteDesk 800 G4 SFF (Rainstorm): the photo of the eBay listing from which we bought the computer, used with the permission of the seller, https://i.ebayimg.com/images/g/UsYAAeSwm7lqAk3f/s-l500.jpg
- Minisforum UM790 Pro: Minisforum, the product image of its store page, https://store.minisforum.com/cdn/shop/files/Compressed_UM790_Pro.png, from https://store.minisforum.com/products/minisforum-um790-pro-mini-pc. The price is from the same page on 2026-10-24.

The memory, power and FP64 figures come from the NVIDIA data sheets: Quadro K420, Quadro K1200, Quadro K6000, Tesla K80, GeForce GTX 1080 and GeForce RTX 3080.

APPENDIX A

All results

Model and setup. Every result in this appendix is for one model, Ternary-Bonsai-2-2B (PTQ1_0), with the pi agent (pi 1.3.0, temperature 0.7, top-p 0.8, top-k 20, a new session for each task). The first two tables ran on the fast test machine: the same model file with llama.cpp on two RTX 3080 cards (How did we test?). They cover the 24 everyday terminal tasks, with 5 runs per task and setup, and the 32 programming and repair tasks, with 3 runs per task and setup. The columns are the five setups. The last table is Kindi 0.1.0 on the Quadro K420 1 GB, with one run of each of 37 tasks. For the 4B, the 9B and the 27B, see Which model for which job? and the Kindi benchmark.

We wrote every task in this appendix for this study: the prompt, the input files and the hidden test that decides if a run passed. Some programming tasks are classic problems that also appear in textbooks and interview practice (for example the longest common subsequence, Dijkstra's shortest path, an LRU cache and Roman numerals). For those tasks the problem is well known, but the prompt and the tests are our own. Each test was proved before use: it fails on the untouched folder and passes on a correct reference solution. The short text jobs, the quiz and the jobs in the breakdown test use fictional messages that we wrote.

Each cell is passes / runs for one task and one setup. The test extension is an experimental pi extension that checks the syntax of files and asks the agent to confirm facts before it stops.

Terminal tasks, the 2B on the fast test machine (5 runs per task and setup)

Task and what it asks	Thinking off, 2,048 output tokens	Thinking off, 8,192	Thinking on, 4,096	Thinking off, 8,192, test extension	Thinking on, 8,192, test extension
All tasks	**37%**	**40%**	**58%**	**42%**	**52%**
largest3 *Find the 3 largest files under a folder; write them largest first.*	0/5	0/5	0/5	0/5	0/5
diskfree *Write the free space of the working filesystem, in 1K blocks.*	1/5	1/5	3/5	0/5	0/5
countpy *Count the lines in all the .py files under a folder (as wc -l counts them).*	0/5	0/5	1/5	0/5	0/5
todos *List every file under a folder that contains the word TODO.*	0/5	1/5	0/5	3/5	0/5
renamejpeg *Rename every .jpeg file in a folder to .jpg and change nothing else.*	3/5	2/5	5/5	1/5	3/5
tarbackup *Make a gzip-compressed tar archive of a folder.*	2/5	3/5	5/5	3/5	4/5
unzipcount *Extract a zip archive and write how many files it held.*	1/5	1/5	2/5	1/5	1/5
execrun *Make a script executable, run it and save its output.*	5/5	5/5	5/5	2/5	5/5
counterrors *Count the ERROR lines in an application log.*	0/5	0/5	0/5	0/5	0/5
topip *Find the client IP address with the most requests in a web log.*	0/5	0/5	0/5	0/5	0/5
dedupenames *Write a list of names once each, sorted.*	5/5	5/5	5/5	5/5	5/5

Task and what it asks	Thinking off, 2,048 output tokens	Thinking off, 8,192	Thinking on, 4,096	Thinking off, 8,192, test extension	Thinking on, 8,192, test extension
emails *Extract the e-mail addresses from free text, without duplicates, sorted.*	5/5	4/5	5/5	5/5	5/5
wordcount *Count the words in a text file (as wc -w counts them).*	0/5	1/5	5/5	0/5	4/5
replaceall *Replace foo with bar in every .txt file under a folder, and only there.*	0/5	0/5	0/5	2/5	0/5
csv2json *Convert a CSV file to a JSON list of objects.*	2/5	5/5	4/5	3/5	5/5
jsonvalue *Write one value from a nested JSON settings file.*	5/5	5/5	5/5	5/5	5/5
sortcsv *Sort a CSV file by its score column, highest first, and keep the header.*	0/5	0/5	3/5	1/5	4/5
editport *Change one line in a settings file and leave every other line as it is.*	5/5	5/5	5/5	5/5	5/5
pyversion *Write the installed version of python3.*	1/5	1/5	5/5	4/5	5/5
gpuname *Write the name of the GPU, as nvidia-smi reports it.*	3/5	3/5	3/5	4/5	4/5
cronline *Write a crontab line that runs a script every day at 02:30.*	0/5	0/5	0/5	0/5	0/5
daysbetween *Write the number of days between two dates.*	1/5	1/5	2/5	0/5	0/5
gitinit *Make a folder a git repository and commit a file with a given message.*	5/5	5/5	5/5	5/5	5/5
hellosh *Write a script that greets the name in an environment variable, with a default.*	0/5	0/5	2/5	1/5	3/5

Programming tasks, the 2B on the fast test machine (3 runs per task and setup), easy to hard

Task and what it asks	Thinking off, 2,048 output tokens	Thinking off, 8,192	Thinking on, 4,096	Thinking off, 8,192, test extension	Thinking on, 8,192, test extension
All tasks	**15%**	**15%**	**10%**	**16%**	**21%**
anagram *Python: group words that are anagrams of each other.*	0/3	1/3	0/3	1/3	0/3
clamp *Python: limit a number to a range; reject a range whose ends are in the wrong order.*	1/3	3/3	2/3	3/3	3/3
fizz *Python: the FizzBuzz list for 1 to n.*	3/3	3/3	2/3	3/3	3/3
flatten *Python: flatten lists nested to any depth, in order.*	2/3	2/3	1/3	0/3	2/3
rle *Python: run-length encode and decode a string.*	0/3	0/3	0/3	0/3	0/3
slugify *JavaScript: turn a title into a lowercase URL slug.*	0/3	0/3	0/3	0/3	0/3
wordfreq *Python: the k most common words in a text, with their counts.*	1/3	1/3	0/3	0/3	1/3
countlines *Shell: a script that prints the number of non-empty lines in a file.*	1/3	0/3	1/3	1/3	2/3
brackets *Python: check that the brackets ()[]{} in a string are balanced.*	2/3	1/3	0/3	0/3	1/3
dedupe *Python: find files with the same content anywhere under a folder.*	0/3	0/3	0/3	0/3	0/3
deepmerge *JavaScript: merge two nested objects, the second one winning.*	0/3	0/3	0/3	0/2	0/3
firstge *Python: binary search for the first element that is not less than x.*	0/3	0/3	0/3	0/3	1/3

Task and what it asks	Thinking off, 2,048 output tokens	Thinking off, 8,192	Thinking on, 4,096	Thinking off, 8,192, test extension	Thinking on, 8,192, test extension
fixavg *Repair: fix a crash on an empty list and a wrong median in a statistics module.*	1/3	1/3	1/3	2/3	2/3
fixcsvsum *Repair: fix a script that adds up the amount column of a CSV file.*	0/3	2/3	0/3	0/3	1/3
fixjs *Repair (JavaScript): fix a word counter that should count lowercase words.*	0/3	0/3	0/3	0/3	0/3
ini *Python: parse INI text into sections of keys and values.*	0/3	0/3	0/3	0/3	0/3
largest *Shell: a script that prints the 3 largest files under a folder.*	0/3	0/3	0/3	0/3	0/3
merge *Python: merge overlapping or touching intervals.*	0/3	0/3	0/3	0/3	0/3
minstack *Python: a stack that also returns its minimum in constant time.*	0/3	0/3	1/3	1/3	1/3
rpn *Python: evaluate a Reverse Polish Notation expression.*	0/3	0/3	0/3	0/3	0/3
spiral *Python: the elements of a matrix in clockwise spiral order.*	0/3	0/3	0/3	0/3	0/3
sumargs *Shell: a script that prints the sum of its integer arguments, with error handling.*	0/3	0/3	0/3	1/3	0/3
dijkstra *Python: the shortest path in a weighted graph, or None.*	1/3	0/3	0/3	0/3	2/3
lcs *Python: one longest common subsequence of two strings.*	2/3	0/3	2/3	3/3	1/3

Task and what it asks	Thinking off, 2,048 output tokens	Thinking off, 8,192	Thinking on, 4,096	Thinking off, 8,192, test extension	Thinking on, 8,192, test extension
bigmul *Python: multiply two very large integers given as strings, without int().*	0/3	0/3	0/3	0/3	0/3
calc *Python: evaluate an arithmetic expression with precedence and parentheses, without eval().*	0/3	0/3	0/3	0/3	0/3
csvline *Python: split one CSV line into fields, with quotes and escaped quotes.*	0/3	0/3	0/3	0/3	0/3
lru *Python: an LRU cache with get and put in constant time.*	0/3	0/3	0/3	0/3	0/3
minwindow *Python: the shortest substring that contains all the characters of another string.*	0/3	0/3	0/3	0/3	0/3
roman *Python: convert numbers to Roman numerals and back, and reject bad input.*	0/3	0/3	0/3	0/3	0/3
topo *Python: a valid order for tasks with dependencies, or None for a cycle.*	0/3	0/3	0/3	0/3	0/3
wordbreak *Python: every way to split a string into words from a dictionary.*	0/3	0/3	0/3	0/3	0/3

The 2B on the Quadro K420 1 GB (Kindi 0.1.0, one run per task)

Task and what it asks	Result	Time
bigmul *Python: multiply two very large integers given as strings, without int().*	wrong	1800 s
calc *Python: evaluate an arithmetic expression with precedence and parentheses, without eval().*	crash	1800 s
csvline *Python: split one CSV line into fields, with quotes and escaped quotes.*	wrong	854 s
lru *Python: an LRU cache with get and put in constant time.*	crash	1800 s

Task and what it asks	Result	Time
minwindow *Python: the shortest substring that contains all the characters of another string.*	crash	1800 s
roman *Python: convert numbers to Roman numerals and back, and reject bad input.*	wrong	184 s
topo *Python: a valid order for tasks with dependencies, or None for a cycle.*	timeout	1102 s
wordbreak *Python: every way to split a string into words from a dictionary.*	timeout	1800 s
clamp *Python: limit a number to a range; reject a range whose ends are in the wrong order.*	pass	139 s
fixavg *Repair: fix a crash on an empty list and a wrong median in a statistics module.*	pass	158 s
fizz *Python: the FizzBuzz list for 1 to n.*	pass	52 s
sumargs *Shell: a script that prints the sum of its integer arguments, with error handling.*	wrong	50 s
wordfreq *Python: the k most common words in a text, with their counts.*	wrong	279 s