

KINDI

A MODERN 2B-PARAMETER LLM AT 20+ TOKENS/S
ON A 2014 NVIDIA QUADRO K420 WITH 1 GB

KINDI FIELD MANUAL · VERSION 0.1.0

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

A private, low-cost AI assistant on an NVIDIA Quadro K420: what it does well, what to avoid, and how to build your own workflows with it.

20.8

tokens per second on the K420 (exact head)

7×

faster than llama.cpp on the same card

1 GB

of GPU memory for a 24,576-token context

0

bytes of your data sent to the cloud

Contents

PREFACE	About this manual	2
CHAPTER 1	Why should you care?	3
CHAPTER 2	How fast can a \$15 GPU be?	6
CHAPTER 3	Meet pi	10
CHAPTER 4	What is an agent?	13
CHAPTER 5	Fast decisions on short text	15
CHAPTER 6	What works, and what to avoid?	19
CHAPTER 7	Can a big job be broken down?	34
CHAPTER 8	Kindi in your own workflows	40
CHAPTER 9	How do you get the best results?	44
CHAPTER 10	Where are the limits?	45
CHAPTER 11	How did we test?	46
CHAPTER 12	Where can you read more?	49
APPENDIX A	All results	50

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.

The manual focuses on a combination of three components: the **pi** harness, which turns a language model into an agent that works in your terminal; the **Ternary-Bonsai-2-2B** model from Prism ML; and **Kindi**, the engine that runs the model on an NVIDIA Quadro K420, a card that you can buy used for about US\$15.

We tested every claim in these pages. We ran 1,269 agent sessions on 56 tasks and recorded each one, and we ran 158 short text jobs such as spam checks and sentiment checks. 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.

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. (Kindi 0.1.0 does not split one request across two cards.)

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. 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 this setup can do for you today, and where its limits are.

CHAPTER 2

How fast can a \$15 GPU be?

Meet the 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 that we wrote for this one chip. 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.

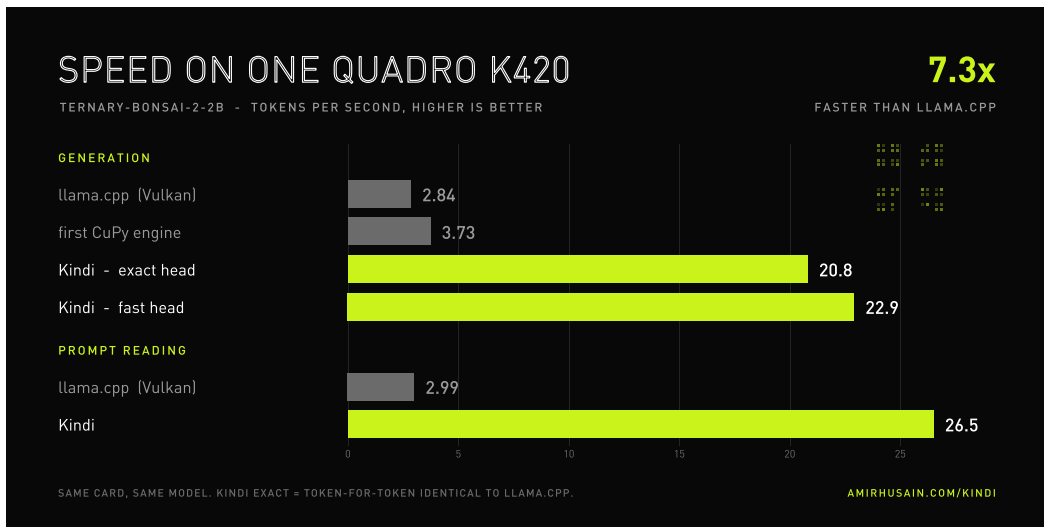


Figure 2-1. Kindi and llama.cpp on the same Quadro K420

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

	llama.cpp (Vulkan)	Kindi (exact head)	Kindi (fast head)
Generation	2.84 tokens/s	20.8 tokens/s	23 tokens/s
Prompt read	2.99 tokens/s	26.5 tokens/s	

Kindi generates text about 7 times faster than llama.cpp and reads a prompt about 9 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. To our knowledge, no other published engine runs a modern language model at this speed on a K420.

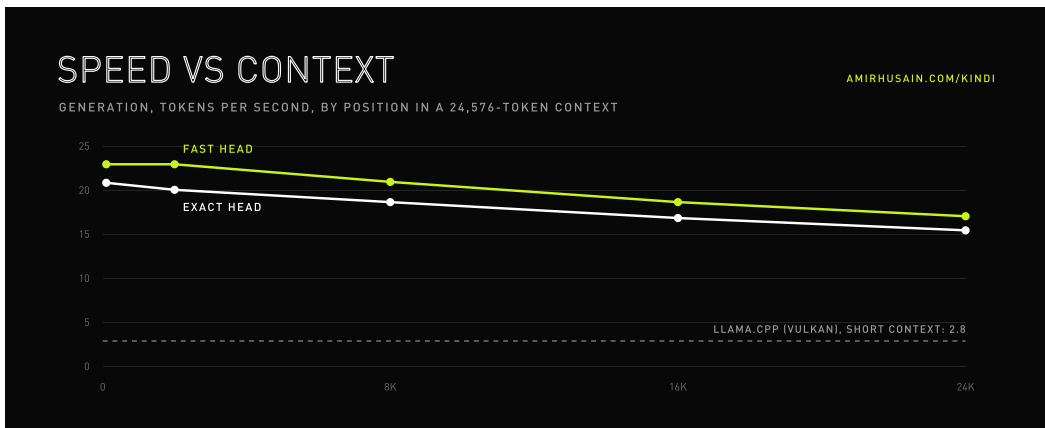


Figure 2-2. *Generation speed by position in a 24,576-token context*

Why does 20 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

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.

```
read contacts.txt
```

I can see the file contains email addresses:

- ann@example.com
- bob.smith@example.org
- zed+test@mail.example.net

Now I need to extract these and write them to emails.txt, sorted alphabetically.

```
write emails.txt
```

```
ann@example.com
bob.smith@example.org
zed+test@mail.example.net
```

The task is complete. I've extracted the email addresses from the contacts.txt file and wrote them to emails.txt in alphabetical order.

Done! The email addresses have been extracted from contacts.txt and saved to emails.txt:

- ann@example.com
- bob.smith@example.org
- zed+test@mail.example.net

All emails are now sorted alphabetically.

```
~/kindi-manual/work
t80 i251 R3.5k CH98.4% 5.7%/25k (auto)
```

```
(kindi-think) bonsai2-2b-k420 • medium
```

Figure 3-1. *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`:

```
{
  "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}
      }]
    }
  }
}
```

Then give requests plenty of time in `~/.pi/agent/settings.json`. The first prompt of an agent holds about 1,000 to 5,000 tokens. Kindi reads it at about 26 tokens per second the first time and then keeps it in its cache.

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

Two of these values come straight from our tests. Set `maxTokens` to 8192, because with 2048 the model often stopped in the middle of a file. Set the thinking level to medium for terminal work, because it raised the success rate of terminal tasks from 40% to 58%.

CHAPTER 4

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:

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"}}
```

The image shows a tmux window with two panes. The left pane contains a terminal session with the following content:

```
v1.3.0
escape interrupt · ctrl+c/ctrl+d clear/exit · / commands · ! bash · ctrl+o more
Press ctrl+o to show full startup help and loaded resources.

Pi can explain its own features and look up its docs. Ask it how to use or extend Pi.

[Context]
~/pi/agent/SYSTEM.md

Warning: tmux extended-keys is off. Modified Enter keys may not work. Add `set -g extended-keys on` to
~/tmux.conf and restart tmux.

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

The user wants me to:
1. Find the value of database.port in a nested settings file
2. Write that value to an answer.txt file
3. Print the answer.txt file contents

I need to first read the file to understand its structure and find the database.port value.

read ./config.json

I can see the config file contains database port information. The database.port value is 5432. Now I
need to write this value to an answer.txt file and print it.

— " Working —
|

~/kindi-manual/work
v52 i87 R1.8k CH95.2% 4.8%/25k (auto) (kindi-think) bonsai2-2b-k420 • medium
```

The right pane is split into two sections:

GPU (nvidia-smi)

GPU	Quadro K420
driver	470.256.02
memory used	875 MiB
memory total	980 MiB
GPU load	91 %
temperature	69
SM clock	875 MHz

KINDI PERFORMANCE

state	generating
answer	55 tokens
speed	19.8 tokens/s
requests	901
cached	3003289 tokens
context	24576 tokens
head exact	
GPU free	104 of 980 MiB
host CPU	100 %
host RAM	23.2 of 47.0 GB

Figure 4-1. A tmux setup: pi with Kindi on the left, the GPU at the top right, Kindi's live performance at the bottom right

CHAPTER 5

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.

Short text jobs on the Quadro K420: share of items correct (Kindi, one try, temperature 0)

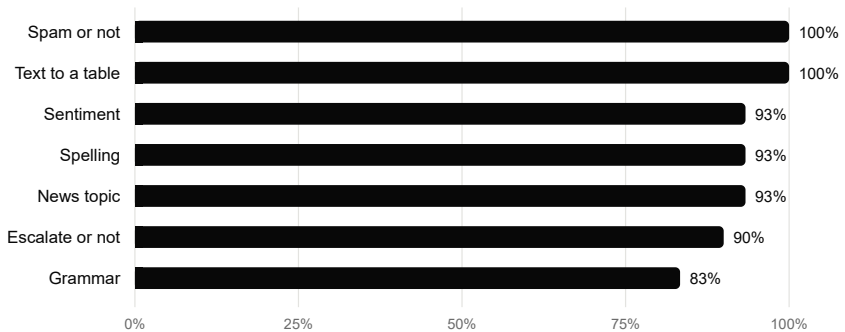


Figure 5-1. Short text jobs on the Quadro K420: the share of items that Kindi got right

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 6

What works, and what to avoid?

This chapter is the heart of the manual. For each type of activity, the chapter tells you what the 2B model does reliably, what it does only sometimes, and what you should not give it at all. 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 we did not run that task on the K420.

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 <i>In the photos folder, rename every file that ends in .jpeg so that it ends in .jpg instead.</i>	2/5	5/5	✗ fail
✓ WORKS	Make a .tar.gz backup of a folder <i>Make a gzip-compressed tar archive named backup.tar.gz that contains the docs folder.</i>	3/5	5/5	✓ pass
✓ WORKS	Make a script executable, run it, save the output <i>Make run.sh executable, run it, and save everything it prints into out.txt.</i>	5/5	5/5	✓ pass
~ SOMETIMES	Unzip an archive and count its files <i>Extract archive.zip into a folder named out, then write the number of files to answer.txt.</i> Do this instead: Let your program run this command. It passes the same hidden test. <pre>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</pre>	1/5	2/5	✓ pass
✗ AVOID	List the files that contain TODO <i>List every file under the src folder that contains the word TODO, sorted, one per line.</i> Do this instead: Let your program run this command. It passes the same hidden test. <pre>grep -rl 'TODO' src sort > answer.txt</pre>	1/5	0/5	✗ fail
✗ AVOID	Count the lines of all .py files <i>How many lines are there in total in all the .py files under src?</i> Do this instead: Let your program run this command. It passes the same hidden test. <pre>find src -name '*.py' -exec cat {} + wc -l > answer.txt</pre>	0/5	1/5	✗ fail
✗ AVOID	Find the 3 largest files <i>Find the 3 largest files under the data folder, largest first, with their sizes.</i> Do this instead: Let your program run this command. It passes the same hidden test. <pre>find data -type f -printf '%s %p\n' sort -rn head -3 > answer.txt</pre>	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 <i>names.txt has one name per line with repeats. Write unique.txt with each name once, sorted.</i>	5/5	5/5	✓ pass
✓ WORKS	Extract e-mail addresses from free text <i>Extract every e-mail address in contacts.txt, remove duplicates, write them to emails.txt sorted.</i>	4/5	5/5	✓ pass
✓ WORKS	Count the words in a document <i>How many words are in essay.txt (as wc -w counts them)?</i>	1/5	5/5	✓ pass
✗ AVOID	Replace a word in many files <i>In every .txt file under the notes folder, replace the word foo with bar.</i> Do this instead: Let your program run this command. It passes the same hidden test. <pre>find notes -name '*.txt' -exec sed -i 's/foo/bar/g' {} +</pre>	0/5	0/5	✗ fail
✗ AVOID	Count the ERROR lines in a log <i>logs/app.log is an application log. How many lines in it have the level ERROR?</i> Do this instead: Let your program run this command. It passes the same hidden test. <pre>awk '\$3=="ERROR"' logs/app.log wc -l > answer.txt</pre>	0/5	0/5	✗ fail
✗ AVOID	Find the busiest client IP in a web log <i>access.log is a web server log. Which client IP address made the most requests?</i> Do this instead: Let your program run this command. It passes the same hidden test. <pre>awk '{print \$1}' access.log sort uniq -c sort -rn head -1 awk '{print \$2}' > answer.txt</pre>	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 <i>config.json is a nested settings file. Write the value of database.port to answer.txt.</i>	5/5	5/5	✓ pass
✓ WORKS	Change one line in a settings file <i>settings.ini has a line port=8080. Change it to port=9090 and leave every other line as it is.</i>	5/5	5/5	✓ pass
✓ WORKS	Convert CSV to JSON <i>Convert people.csv to people.json: a JSON list with one object per row.</i>	5/5	4/5	✓ pass
~ SOMETIMES	Sort a CSV file by a column <i>scores.csv has a header row. Write sorted.csv with the rows sorted by score, highest first.</i> Do this instead: Let your program run this command. It passes the same hidden test. <pre>(head -1 scores.csv; tail -n +2 scores.csv sort -t, -k2,2nr) > sorted.csv</pre>	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"}}
```

```
read config.json
```

I can see the **config.json** file contains a JSON object with a "database" key that has a "port" value of 5432. Now I need to write this value to **answer.txt** and print it.

```
write answer.txt
```

```
5432
```

The file has been written successfully. Now I need to print the **answer.txt** file to show the user the result.

```
read answer.txt
```

The **answer.txt** file contains the value 5432, which is the **database.port** value from the **config.json** file. The task is complete.

The value of **database.port** is 5432.

answer.txt contains:

```
5432
```

```
~/kindi-manual/work  
t201 t280 R4.8k CH95.0% 5.9%/25k (auto)
```

```
(kindi-think) bonsai2-2b-k420 • medium
```

Figure 6-1. *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 <i>Make the project folder a git repository and commit README.md with the message: initial.</i>	5/5	5/5	✓ pass
✓ WORKS	Report the installed Python version <i>Which version of python3 is installed? Write only the version number to answer.txt.</i>	1/5	5/5	✓ pass
~ SOMETIMES	Report the GPU name <i>What is the name of the first NVIDIA GPU, as nvidia-smi reports it?</i> Do this instead: Let your program run this command. It passes the same hidden test. <code>nvidia-smi --query-gpu=name --format=csv,noheader -i 0 > answer.txt</code>	3/5	3/5	✓ pass
~ SOMETIMES	Report the free disk space <i>How much space is free on the filesystem that holds the working folder (df -k)?</i> Do this instead: Let your program run this command. It passes the same hidden test. <code>df -k --output=avail . tail -1 tr -d ' ' > answer.txt</code>	1/5	3/5	✗ fail
~ SOMETIMES	Count the days between two dates <i>How many days are there from 2026-01-15 to 2026-10-18?</i> Do this instead: Let your program run this command. It passes the same hidden test. <code>python3 -c "from datetime import date; print((date(2026, 10, 18) - date(2026, 1, 15)).days)" > answer.txt</code>	1/5	2/5	✗ fail
✗ AVOID	Write a crontab line <i>Write one crontab line to cron.txt that runs /opt/backup.sh every day at 02:30.</i> Do this instead: Let your program run this command. It passes the same hidden test. <code>echo '30 2 * * * /opt/backup.sh' > cron.txt</code>	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
```

```
[master (root-commit) 2835512] initial
1 file changed, 1 insertion(+)
create mode 100644 README.md
```

Took 0.0s

The commit was successful. Now I need to show the git log to display the commit history.

```
$ cd project && git log --oneline
```

```
2835512 initial
```

Took 0.0s

The git log shows the commit history with one commit. Let me provide a summary of what was done.

The project has been set up with a git repository. Here's what was done:

1. Created a git repository in the `project` directory
2. Added the `README.md` file
3. Committed the file with the message "initial"
4. The git log shows the commit history

```
2835512 initial
```

```
~/kindi-manual/work
+622 +691 R15k CH99.0% 9.7%/25k (auto)
```

```
(kindi-think) bonsai2-2b-k420 • medium
```

Figure 6-2. *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 <i>Write hello.sh that prints Hello, NAME! where NAME comes from the environment, or world.</i> Do this instead: Let your program run this command. It passes the same hidden test. <pre>printf '#!/bin/bash\nnecho "Hello, \${NAME:-world}!"\n' > hello.sh</pre>	0/5	2/5	✗ fail
~ SOMETIMES	A script that counts non-empty lines <i>Create countlines.sh that prints the number of non-empty lines in a file, or an error.</i> 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 <i>Create sumargs.sh that prints the sum of its integer arguments, with an error for bad input.</i> 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 <i>Create largest.sh that prints the 3 largest files under a folder, with sizes.</i> Do this instead: Put the command from the files table in the script: <code>find DIR -type f -printf '%s %p\n' sort -rn head -3</code> . 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 <code>anagram.py</code> with <code>group_anagrams(words)</code> , 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 <code>first_ge.py</code> with <code>first_ge(arr, x)</code> in $O(\log n)$, without <code>bisect</code> . Do this instead: In your own code, use <code>bisect.bisect_left</code> 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 <code>rle.py</code> with <code>encode('aaab') == 'a3b1'</code> and <code>decode</code> . Do this instead: Use <code>itertools.groupby</code> 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 <code>merge.py</code> with <code>merge(intervals)</code> . 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 <code>rpn.py</code> with <code>rpn('3 4 + 2 *') == 14</code> and errors for bad input. Do this instead: Write it yourself: a list as the stack and one loop over the tokens. The <code>operator</code> 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 <i>Create spiral.py with spiral(matrix).</i> 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 <i>Create ini.py with parse_ini(text) that returns sections of keys and values.</i> 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 <i>Create dedupe.py with dedupe_files(folder).</i> 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 <i>report.py should print the total of the amount column of sales.csv, but it crashes. Fix it.</i> 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 <i>stats.py: mean([]) crashes and median() is wrong for even lengths. Fix it.</i> 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 <i>counter.js should count lowercase words. It has bugs. Fix them.</i> 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 <i>Create slugify.js that exports slugify(s).</i> 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 <i>Create deepmerge.js that exports deepMerge(a, b).</i> 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 <i>counter.js should count lowercase words. It has bugs. Fix them.</i> 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 <code>lcs.py</code> with <code>lcs(a, b)</code> , 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 <code>dijkstra.py</code> with <code>shortest_path(graph, start, goal)</code> . 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 <code>topo.py</code> with <code>order(n, deps)</code> , smallest order first, None for a cycle. Do this instead: Use <code>graphlib.TopologicalSorter</code> 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 <code>roman.py</code> with <code>to_roman(n)</code> and <code>from_roman(s)</code> , 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 <i>Create calc.py with evaluate(s) for + - * / and parentheses.</i> 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 <i>Create csvline.py with parse_line(line), with quoted fields.</i> Do this instead: Use the csv module of the Python standard library: <code>next(csv.reader([line]))</code> parses one line. In our tests, 0 of 15 tries passed.	0/3	0/3	✗ fail
✗ AVOID	A least-recently-used cache <i>Create lru.py with an LRUcache class.</i> Do this instead: Use <code>functools.lru_cache</code> , or <code>collections.OrderedDict</code> with <code>move_to_end</code> . In our tests, 0 of 15 tries passed.	0/3	0/3	✗ fail
✗ AVOID	Split text into dictionary words <i>Create wordbreak.py with sentences(s, words).</i> Do this instead: Write it yourself as a recursive function with <code>functools.cache</code> . In our tests, 0 of 15 tries passed.	0/3	0/3	✗ fail
✗ AVOID	Multiply big numbers given as text <i>Create bigmul.py with multiply(a, b), digit by digit.</i> Do this instead: In Python, <code>int(a) * int(b)</code> 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 <i>Create minwindow.py with min_window(s, t) in linear time.</i> Do this instead: Write it yourself with two pointers and a <code>collections.Counter</code> . In our tests, 0 of 15 tries passed.	0/3	0/3	✗ fail

What is the pattern?

Share of sessions that passed, by activity (thinking off and on together)

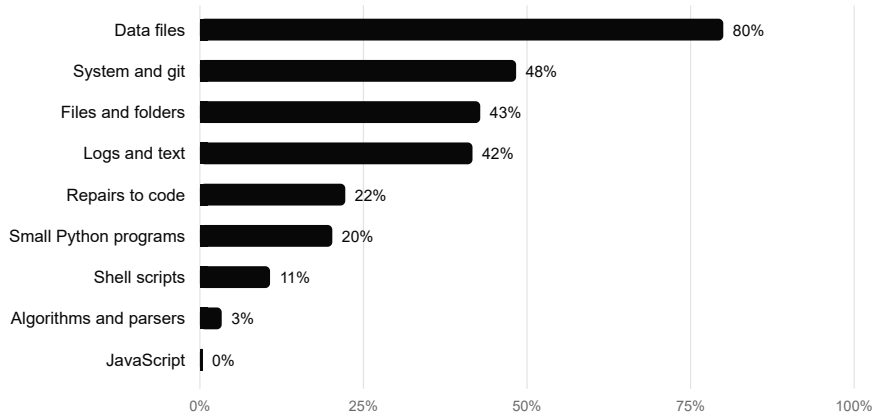


Figure 6-3. *Success rate by type of activity*

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 7

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

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 8

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 9

How do you get the best results?

- **Expect small tasks to take seconds.** On the K420, with thinking on, each terminal task that passed took between 20 seconds and 2.5 minutes from start to end.
- **The first prompt is slow; the next ones are fast.** Kindi saves the start of the agent's prompt on disk, so the second session starts in a fraction of a second.
- **Use thinking for terminal work.** In our terminal tests, thinking raised the success rate from 40% to 58%.
- **For code, try both.** In our programming tests, thinking alone lowered the success rate from 15% to 10%.
- **Set the output limit to 8,192 tokens.** With 2,048, the model often stopped in the middle of a file.
- **Ask for the answer in a file.** In 32% of the 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 this model.
- **Keep files small.** The context is 24,576 tokens, and a large file fills it fast.

CHAPTER 10

Where are the limits?

Here is exactly what this setup is:

- Bonsai 2 2B is a 2-billion-parameter model, and it does not replace a large cloud model on hard problems.
- Of our 10 hard programming tasks, the model solved only two, and only some of the time. The model did not solve any of the terminal tasks that need several connected steps in one prompt, such as count, then sort, then select.
- The model can say that a job is complete when it is not. Always check.
- One Kindi server answers one request at a time, so several agents on one card wait for each other.
- Kindi 0.1.0 runs one model (Ternary-Bonsai-2-2B) on the Quadro K420. Other Kepler cards are not tested.

These limits are real. So is the result: a private assistant that does useful small jobs, on a card that costs about US\$15.

CHAPTER 11

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 all Kindi numbers in this manual come from the K420.
- **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

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. The transcripts of every session will follow with a separate report.

CHAPTER 12

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-2-2B.** The model, from Prism ML: <https://huggingface.co/prism-ml/Ternary-Bonsai-2-2B-gguf>.
- **llama.cpp.** The reference engine that Kindi's output matches: <https://github.com/ggml-org/llama.cpp>.
- **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>.

APPENDIX A

All results

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](#steps) 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. It is not published yet.

Terminal tasks (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 <i>Find the 3 largest files under a folder; write them largest first.</i>	0/5	0/5	0/5	0/5	0/5
diskfree <i>Write the free space of the working filesystem, in 1K blocks.</i>	1/5	1/5	3/5	0/5	0/5
countpy <i>Count the lines in all the .py files under a folder (as wc -l counts them).</i>	0/5	0/5	1/5	0/5	0/5

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
todos <i>List every file under a folder that contains the word TODO.</i>	0/5	1/5	0/5	3/5	0/5
renamejpeg <i>Rename every .jpeg file in a folder to .jpg and change nothing else.</i>	3/5	2/5	5/5	1/5	3/5
tarbackup <i>Make a gzip-compressed tar archive of a folder.</i>	2/5	3/5	5/5	3/5	4/5
unzipcount <i>Extract a zip archive and write how many files it held.</i>	1/5	1/5	2/5	1/5	1/5
execrun <i>Make a script executable, run it and save its output.</i>	5/5	5/5	5/5	2/5	5/5
counterrors <i>Count the ERROR lines in an application log.</i>	0/5	0/5	0/5	0/5	0/5
topip <i>Find the client IP address with the most requests in a web log.</i>	0/5	0/5	0/5	0/5	0/5
dedupenames <i>Write a list of names once each, sorted.</i>	5/5	5/5	5/5	5/5	5/5
emails <i>Extract the e-mail addresses from free text, without duplicates, sorted.</i>	5/5	4/5	5/5	5/5	5/5
wordcount <i>Count the words in a text file (as wc -w counts them).</i>	0/5	1/5	5/5	0/5	4/5
replaceall <i>Replace foo with bar in every .txt file under a folder, and only there.</i>	0/5	0/5	0/5	2/5	0/5
csv2json <i>Convert a CSV file to a JSON list of objects.</i>	2/5	5/5	4/5	3/5	5/5
jsonvalue <i>Write one value from a nested JSON settings file.</i>	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
sortcsv <i>Sort a CSV file by its score column, highest first, and keep the header.</i>	0/5	0/5	3/5	1/5	4/5
editport <i>Change one line in a settings file and leave every other line as it is.</i>	5/5	5/5	5/5	5/5	5/5
pyversion <i>Write the installed version of python3.</i>	1/5	1/5	5/5	4/5	5/5
gpu name <i>Write the name of the GPU, as nvidia-smi reports it.</i>	3/5	3/5	3/5	4/5	4/5
cronline <i>Write a crontab line that runs a script every day at 02:30.</i>	0/5	0/5	0/5	0/5	0/5
daysbetween <i>Write the number of days between two dates.</i>	1/5	1/5	2/5	0/5	0/5
gitinit <i>Make a folder a git repository and commit a file with a given message.</i>	5/5	5/5	5/5	5/5	5/5
hellosk <i>Write a script that greets the name in an environment variable, with a default.</i>	0/5	0/5	2/5	1/5	3/5

Programming tasks (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 <i>Python: group words that are anagrams of each other.</i>	0/3	1/3	0/3	1/3	0/3
clamp <i>Python: limit a number to a range; reject a range whose ends are in the wrong order.</i>	1/3	3/3	2/3	3/3	3/3

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
fizz <i>Python: the FizzBuzz list for 1 to n.</i>	3/3	3/3	2/3	3/3	3/3
flatten <i>Python: flatten lists nested to any depth, in order.</i>	2/3	2/3	1/3	0/3	2/3
rle <i>Python: run-length encode and decode a string.</i>	0/3	0/3	0/3	0/3	0/3
slugify <i>JavaScript: turn a title into a lowercase URL slug.</i>	0/3	0/3	0/3	0/3	0/3
wordfreq <i>Python: the k most common words in a text, with their counts.</i>	1/3	1/3	0/3	0/3	1/3
countlines <i>Shell: a script that prints the number of non-empty lines in a file.</i>	1/3	0/3	1/3	1/3	2/3
brackets <i>Python: check that the brackets ()[]{} in a string are balanced.</i>	2/3	1/3	0/3	0/3	1/3
dedupe <i>Python: find files with the same content anywhere under a folder.</i>	0/3	0/3	0/3	0/3	0/3
deepmerge <i>JavaScript: merge two nested objects, the second one winning.</i>	0/3	0/3	0/3	0/2	0/3
firstge <i>Python: binary search for the first element that is not less than x.</i>	0/3	0/3	0/3	0/3	1/3
fixavg <i>Repair: fix a crash on an empty list and a wrong median in a statistics module.</i>	1/3	1/3	1/3	2/3	2/3
fixcsvsum <i>Repair: fix a script that adds up the amount column of a CSV file.</i>	0/3	2/3	0/3	0/3	1/3
fixjs <i>Repair (JavaScript): fix a word counter that should count lowercase words.</i>	0/3	0/3	0/3	0/3	0/3

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
ini <i>Python: parse INI text into sections of keys and values.</i>	0/3	0/3	0/3	0/3	0/3
largest <i>Shell: a script that prints the 3 largest files under a folder.</i>	0/3	0/3	0/3	0/3	0/3
merge <i>Python: merge overlapping or touching intervals.</i>	0/3	0/3	0/3	0/3	0/3
minstack <i>Python: a stack that also returns its minimum in constant time.</i>	0/3	0/3	1/3	1/3	1/3
rpn <i>Python: evaluate a Reverse Polish Notation expression.</i>	0/3	0/3	0/3	0/3	0/3
spiral <i>Python: the elements of a matrix in clockwise spiral order.</i>	0/3	0/3	0/3	0/3	0/3
sumargs <i>Shell: a script that prints the sum of its integer arguments, with error handling.</i>	0/3	0/3	0/3	1/3	0/3
dijkstra <i>Python: the shortest path in a weighted graph, or None.</i>	1/3	0/3	0/3	0/3	2/3
lcs <i>Python: one longest common subsequence of two strings.</i>	2/3	0/3	2/3	3/3	1/3
bigmul <i>Python: multiply two very large integers given as strings, without int().</i>	0/3	0/3	0/3	0/3	0/3
calc <i>Python: evaluate an arithmetic expression with precedence and parentheses, without eval().</i>	0/3	0/3	0/3	0/3	0/3
csvline <i>Python: split one CSV line into fields, with quotes and escaped quotes.</i>	0/3	0/3	0/3	0/3	0/3

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
lru <i>Python: an LRU cache with get and put in constant time.</i>	0/3	0/3	0/3	0/3	0/3
minwindow <i>Python: the shortest substring that contains all the characters of another string.</i>	0/3	0/3	0/3	0/3	0/3
roman <i>Python: convert numbers to Roman numerals and back, and reject bad input.</i>	0/3	0/3	0/3	0/3	0/3
topo <i>Python: a valid order for tasks with dependencies, or None for a cycle.</i>	0/3	0/3	0/3	0/3	0/3
wordbreak <i>Python: every way to split a string into words from a dictionary.</i>	0/3	0/3	0/3	0/3	0/3

On the Quadro K420 (Kindi, one run per task)

Task and what it asks	Result	Time
bigmul <i>Python: multiply two very large integers given as strings, without int().</i>	wrong	1800 s
calc <i>Python: evaluate an arithmetic expression with precedence and parentheses, without eval().</i>	crash	1800 s
csvline <i>Python: split one CSV line into fields, with quotes and escaped quotes.</i>	wrong	854 s
lru <i>Python: an LRU cache with get and put in constant time.</i>	crash	1800 s
minwindow <i>Python: the shortest substring that contains all the characters of another string.</i>	crash	1800 s
roman <i>Python: convert numbers to Roman numerals and back, and reject bad input.</i>	wrong	184 s
topo <i>Python: a valid order for tasks with dependencies, or None for a cycle.</i>	timeout	1102 s
wordbreak <i>Python: every way to split a string into words from a dictionary.</i>	timeout	1800 s

Task and what it asks	Result	Time
clamp <i>Python: limit a number to a range; reject a range whose ends are in the wrong order.</i>	pass	139 s
fixavg <i>Repair: fix a crash on an empty list and a wrong median in a statistics module.</i>	pass	158 s
fizz <i>Python: the FizzBuzz list for 1 to n.</i>	pass	52 s
sumargs <i>Shell: a script that prints the sum of its integer arguments, with error handling.</i>	wrong	50 s
wordfreq <i>Python: the k most common words in a text, with their counts.</i>	wrong	279 s