

Shopify — 30-minute conversation

Recent work, impact, and how you use AI · prep sheet

What this is. The invitation says: come prepared to talk about projects you have been working on, your impact, and *how you're using AI*. That last clause is not small talk. Shopify is the company whose CEO publicly declared reflexive AI usage a baseline expectation and made it part of performance review. In a 30-minute call, that question is a screen.

The thesis: two thirds of this call is a normal “walk me through your work” conversation, and you should prepare it like one. But the AI third is where candidates separate. Almost everyone will say “I use Copilot.” Section 3 is how to give an answer that sounds like you belong at Shopify specifically.

Budget the 30 minutes

Minute	What should be happening
0–3	Intros, how the process works. Short.
3–8	Your arc. Where you are now and how you got there.
8–20	One project, in depth. Problem, decisions, impact, numbers.
20–26	AI. How you actually work with it. Be specific.
26–30	Your questions.

Half an hour goes fast. Rehearse the project story to about eight minutes spoken — if it runs to fifteen, you will lose the AI section, which is the part they flagged in advance.

1. The project story

Shopify's distinctive round later in the loop is the **Life Story** interview — a chronological walk through your career focused on *why* you made each decision, not a rehearsed highlight reel. This 30-minute call is adjacent to that, and the same instinct applies: they are buying judgement and self-awareness, not polish.

Structure for the one deep project

- 1. **The situation in one sentence, with a number in it.** Not “I worked on the checkout service” but “checkout was timing out for about 2% of sessions during peak, and that 2% was worth roughly \$X.”
- 2. **Why it was hard.** The constraint. If there was no constraint, it was not a story worth telling.
- 3. **What you decided, and what you rejected.** Name the alternative. Say what would have made you choose it.
- 4. **What you shipped and what changed.** Measured, before and after.
- 5. **What you would do differently.** Volunteer this. Shopify explicitly evaluates self-awareness, and candidates who only tell wins score worse than candidates who tell one honest failure well.

Impact — say it in their units

Shopify's cultural centre of gravity is **merchant obsession**: the person at the other end of the software is a small business owner who loses a real sale when your code is slow or wrong. Translate your impact into a user's experience wherever you can.

Weak	Strong
“Improved performance by 40%”	“Cut p99 from 1.2s to 700ms, which took the timeout rate at peak from 2% to near zero — those were abandoned carts”
“Refactored the service”	“The old design meant every schema change needed a coordinated deploy; after, one team could ship without blocking three others”
“Led the migration”	“I owned the cutover plan. We moved 100% of traffic with no rollback, and the thing I got wrong was underestimating backfill time by a week”

Have **three** projects loaded, not one: the technically deepest, the one with the clearest business impact, and the one that went wrong. They will likely only ask for one, but you want to choose rather than scramble.

2. Shopify's AI position

Context you should know before answering the AI question, because it tells you what a good answer sounds like there.

- In **April 2025** CEO Tobi Lütke published an internal memo declaring **“reflexive AI usage is now a baseline expectation at Shopify.”** It became a genre — Box, Fiverr and others published their own versions within weeks.
- The memo made AI usage part of **performance and peer review**, and required teams to demonstrate why they *could not* get what they wanted done using AI before asking for more headcount. The press framing was “no new hires unless AI can't do the job.”
- It was not a standing start. VP & Head of Engineering **Farhan Thawar** brought GitHub Copilot in during **late 2021**, a year before ChatGPT, and hit 80% adoption quickly.
- Thawar estimates the engineering org is roughly **20% more productive** — and is explicit that this does *not* show up in lines of code or PR count, which are gameable. It shows up as faster prototyping and **exploring ten approaches instead of two**. Their preferred progress signal is weekly demos.
- They ordered **1,500 Cursor licences, then another 1,500**. The fastest-growing user groups are **support and revenue**, not engineering.
- Architecturally they **standardised the infrastructure, not the tool**: an internal LLM proxy routes every request through one gateway, so engineers can use Claude Code, Copilot or Cursor while the company keeps cost control, usage analytics and model flexibility.

The two ideas to actually borrow

1. Comprehension debt. Thawar names this as the number one long-term risk: if engineers stop thinking deeply, they lose understanding of their own systems and cannot maintain them when things break. His guardrail is that engineers must understand the system **two to three layers below** where they are working, and use AI to *accelerate* learning rather than replace it.

2. Agentic harnesses. Their stated 2026 bet is orchestration — either parallel execution (many agents working at once, human reviews and merges) or sequential critique loops (long thinking sessions with multi-model interrogation). Thawar's line: *if you don't figure out how to harness agents in 2026, you'll be behind.*

If you use the phrase **comprehension debt** and mean it, you are speaking their language using their own framing. It is the single highest-leverage two words in this call.

3. Answering “how are you using AI?”

What a weak answer sounds like

- “I use Copilot for autocomplete.” (Everyone does. This is 2021.)
- “I use ChatGPT when I'm stuck.” (No workflow, no judgement.)
- “I'm cautious about AI, it hallucinates.” (True and irrelevant; at Shopify this reads as not having done the work.)
- Uncritical maximalism — “I let it write everything.” They will immediately probe how you verify, and comprehension debt is their own stated fear.

The shape of a strong answer

1. **Name the workflow, not the tool.** What you actually delegate: scaffolding, test generation, unfamiliar-API exploration, mechanical refactors across many files, first-draft migration scripts, reading a codebase you did not write.
2. **Say what you keep.** Architecture, data model, the failure modes, anything where being subtly wrong is expensive. The contrast is the substance of the answer.
3. **Describe how you verify.** Tests first, small diffs, reading the generated code before running it, treating agent output as a colleague's PR rather than as an oracle. Concrete beats principled.
4. **Give one number or one before/after.** “That migration would have been two days of mechanical edits; it was about three hours plus review.” Or “I now try three designs before picking one, where I used to try one.”
5. **Name the risk unprompted** — comprehension debt — and say your guardrail against it.
6. **Say where you are going next.** Agent orchestration, running several in parallel, building your own harness or tooling. This is exactly their 2026 thesis.

A worked example you can adapt

“My default now is an agent in the loop rather than autocomplete. For anything mechanical — a refactor across forty files, a first pass at tests, exploring an API I haven't used — I'll hand it off, then review the diff the way I'd review a colleague's PR. What I don't hand off is the data model or anything where being subtly wrong is expensive, because I want to have actually reasoned about it.

The concrete win is breadth: I'll now try three approaches to something before committing, where before I'd pick one and hope. The concrete risk is what Shopify's own engineering leadership calls comprehension debt — you can ship code you don't understand. My guardrail is that I won't merge something I

couldn't debug at 3am, so if I can't explain what the generated code does, that's the signal to slow down and read it properly."

Then have **one specific story** ready where AI genuinely changed the outcome of a piece of work — and, if you have it, one where you tried it and it was the wrong tool. The second story is more credible than the first.

They will likely ask a follow-up: **"where has it not worked?"** Have an answer. Good ones: anything needing context that is not in the repo, debugging genuinely novel failures, and code where the cost of a plausible-looking mistake is high. Having a real boundary makes the enthusiasm credible.

4. The rest of the loop

Worth knowing now, because they may describe the process and you should sound unsurprised. Shopify's loop runs about 3–5 weeks:

Stage	Shape
Recruiter screen	30–45 min. Background, mission fit, logistics.
Life Story	60 min. Chronological career walk-through. Widely described as the make-or-break round. Authenticity over polish; no STAR recitation.
Technical challenge	60–90 min on CoderPad. Practical problems, not abstract puzzles.
Pair programming	60–90 min building a small feature with an engineer. Your own IDE, and AI tools are explicitly encouraged .
Deep dive + design	~60 min: a past project, plus a system design question in Shopify's domain.
Offer + team match	Mutual fit conversation.

The pairing round is unusual and worth internalising now. Shopify wants you to use Cursor or Copilot during it — but the thing being scored is whether you explain your reasoning and verify the output, or let the model drive the logic. The framing that works: *AI handles the boilerplate, you handle the architecture.*

5. The technical rounds

Shopify's coding questions are medium difficulty and deliberately practical — framed around commerce and simulation rather than abstract algorithms. Reviewers weight **code quality and edge cases** over optimal Big O.

Problem	Freq	Pattern
Walking Robot Simulation	100	Simulation + hash set of obstacles
LRU Cache	84	Design, hash + linked list
Design In-Memory File System	60	Trie / nested dict, Hard
Encode and Decode TinyURL	60	Design, hashing
Paint House	60	Small DP
Design a Text Editor	52	Two stacks, Hard

Also reported: an *Optimal Shipping Rate Calculator*, a *Mars Rover simulation*, and building a URL shortener end to end. The through-line is obvious — **model a small world and make it behave correctly**.

Walking Robot Simulation — the archetype

```
def robotSim(commands, obstacles):
    # directions in turning order: N, E, S, W
    DIRS = [(0, 1), (1, 0), (0, -1), (-1, 0)]
    obs = {tuple(o) for o in obstacles}      # set, not list: O(1) lookup
    x = y = di = 0
    best = 0
    for c in commands:
        if c == -1:
            di = (di + 1) % 4                # turn right
        elif c == -2:
            di = (di - 1) % 4                # turn left
        else:
            dx, dy = DIRS[di]
            for _ in range(c):
                if (x + dx, y + dy) in obs:
                    break                    # blocked: stop, keep facing
                x, y = x + dx, y + dy
            best = max(best, x * x + y * y)  # max squared distance EVER
    return best
```

Three traps, all of which are the actual question: obstacles must be a **set** or it is $O(n \cdot k)$; you must step **one square at a time** to detect a blocker mid-move; and the answer is the maximum distance reached at *any* point, not the final position. Ordering the directions so that turning is $\pm 1 \bmod 4$ is the tidy touch.

Encode and Decode TinyURL

```
import random, string
```

```

class Codec:
    def __init__(self):
        self.to_long = {}
        self.to_short = {}
        self.chars = string.ascii_letters + string.digits

    def encode(self, longUrl: str) -> str:
        if longUrl in self.to_short: # idempotent: same in, same out
            return self.to_short[longUrl]
        while True:
            code = "".join(random.choices(self.chars, k=6))
            if code not in self.to_long: # collision check
                break
            self.to_long[code] = longUrl
            self.to_short[longUrl] = code
        return "http://tinyurl.com/" + code

    def decode(self, shortUrl: str) -> str:
        return self.to_long[shortUrl.rsplit("/", 1)[-1]]

```

Expect the follow-up to be the real interview: 62^6 is about 57 billion codes, so how do you avoid collisions at scale (counter + base62, or a hash with a collision check), how do you shard, and how would you handle custom aliases and expiry? That is a system design question wearing a coding question's clothes.

Design In-Memory File System

```

class Node:
    def __init__(self):
        self.children = {} # name -> Node
        self.content = ""
        self.is_file = False # explicit: an empty dir is not a file

class FileSystem:
    def __init__(self):
        self.root = Node()

    def _walk(self, path, create=False):
        node = self.root
        for part in path.split("/"):
            if not part: # "" from the leading slash
                continue
            if part not in node.children:
                if not create:
                    return None
                node.children[part] = Node()
            node = node.children[part]
        return node

    def ls(self, path):
        node = self._walk(path)
        if node.is_file:
            return [path.rsplit("/", 1)[-1]] # a file: its own name
        return sorted(node.children) # a directory

    def mkdir(self, path):
        self._walk(path, create=True)

    def addContentToFile(self, path, content):

```

```
node = self._walk(path, create=True)
node.is_file = True
node.content += content

def readContentFromFile(self, path):
    return self._walk(path).content
```

Files and directories are the *same* node type — modelling them as two classes doubles the code for no gain. But keep `is_file` explicit rather than inferring “directory” from having children: an empty directory has no children either, and `ls("/")` on a fresh filesystem must return `[]` rather than a phantom entry. That edge case is what the question is actually about.

System design prompts at Shopify

- **Handle a 100x traffic spike** (Black Friday / Cyber Monday). The best answers talk about shedding load gracefully, queueing checkouts rather than dropping them, pre-warming, and what you sacrifice first.
- **Multi-region resilient checkout.** The consistency question is unavoidable: you cannot double-charge, and you cannot lose an order.
- **Inventory management.** The classic oversell problem — reservations, idempotency, and whether you allow overselling and reconcile, or lock and lose throughput.
- **Background job processor with retries**, or a **shopping cart state machine** — these show up in the pairing round as build-it-iteratively tasks.

Merchant obsession is the tiebreaker. Frame design decisions around what protects a small business owner from losing a sale, not just what satisfies a latency budget. A slightly imperfect design you can defend in those terms beats a theoretically better one you cannot.

Questions to ask, and what to avoid

Ask

- “Thawar has talked about comprehension debt as the main long-term risk of going AI-first. How does that show up in practice — has it changed how you do code review?”
- “You measure AI impact with weekly demos rather than PR counts. What convinced you the usual metrics were the wrong ones?”
- “What does the first ninety days look like for this role, and what would make you say afterwards that it clearly worked?”

The first two are drawn from Shopify's own public engineering commentary. They land because they show you engaged with how the company actually works rather than reading a careers page.

Avoid

- Reciting your resume chronologically without being asked. Save it for the Life Story round, where it is the format.
- Impact claims with no number and no user attached.
- Hedging on AI. A thoughtful, specific position — even a partly sceptical one — beats a vague positive one. Vagueness is the failure mode here, not disagreement.
- Running the project story long. If you are at fifteen minutes and have not talked about AI, stop and pivot yourself.

If you prepare one thing, prepare the AI answer: a specific workflow, a clear line between what you delegate and what you keep, one number, and comprehension debt named out loud. That is the answer this call was scheduled to hear.