

Alibaba releases Qwen3.5 with a 397-billion-parameter open-weights model

Source copy kept by wheresthe.ai for the story "Alibaba releases Qwen3.5 with a 397-billion-parameter open-weights model" (<https://wheresthe.ai/d/cn-alibaba-qwen-3-5-open-weights-2026-02/>).

Source: <https://qwen.ai/blog?id=qwen3.5>

Text saved 18 Oct 2026 (the page text through a reader, because the site refused a direct fetch).

SHA-256 of the saved text: 923d4d5736650284b82a9b5084e30917adac46bbc0daad238988bf526d6d8108

QWEN CHATGitHubHugging FaceModelScopeDISCORD

We are delighted to announce the official release of **Qwen3.5**, introducing the open-weight of the first model in the Qwen3.5 series, namely **Qwen3.5-397B-A17B**. As a native vision-language model, Qwen3.5-397B-A17B demonstrates outstanding results across a full range of benchmark evaluations, including reasoning, coding, agent capabilities, and multimodal understanding, empowering developers and enterprises to achieve significantly greater productivity. Built on an innovative hybrid architecture that fuses linear attention (via Gated Delta Networks) with a sparse mixture-of-experts, the model attains remarkable inference efficiency: although it comprises 397 billion total parameters, just 17 billion are activated per forward pass, optimizing both speed and cost without sacrificing capability. We have also expanded our language and dialect support from 119 to 201, providing broader accessibility and enhanced support to users around the world.

Qwen3.5-Plus is the hosted model available via Alibaba Cloud Model Studio, featuring:
* a 1M context window by default
* official built-in tools and adaptive tool use

Performance

Below we present the comprehensive evaluation of our models against frontier models in a wide range of evaluation tasks, covering different tasks and modalities.

Language

GPT5.2 Claude 4.5 Opus Gemini-3 Pro Qwen3-Max-Thinking K2.5-1T-A32B Qwen3.5-397B-A17B ---
--- --- --- --- --- --- Knowledge MMLU-Pro 87.4 89.5 89.8 85.7 87.1 87.8 MMLU-Redux 95.0
95.6 95.9 92.8 94.5 94.9 SuperGPQA 67.9 70.6 74.0 67.3 69.2 70.4 C-Eval 90.5 92.2 93.4
93.7 94.0 93.0 Instruction Following IFEval 94.8 90.9 93.5 93.4 93.9 92.6 IFBench 75.4 58.0
70.4 70.9 70.2 76.5 MultiChallenge 57.9 54.2 64.2 63.3 62.7 67.6 Long Context AA-LCR 72.7
74.0 70.7 68.7 70.0 68.7 LongBench v2 54.5 64.4 68.2 60.6 61.0 63.2 STEM GPQA 92.4
87.0 91.9 87.4 87.6 88.4 HLE 35.5 30.8 37.5 30.2 30.1 28.7 HLE-Verified ¹ 43.3 38.8 48 37.6
-- 37.6 Reasoning LiveCodeBench v6 87.7 84.8 90.7 85.9 85.0 83.6 HMMT Feb 25 99.4 92.9
97.3 98.0 95.4 94.8 HMMT Nov 25 100 93.3 93.3 94.7 91.1 92.7 IMOAnswerBench 86.3 84.0
83.3 83.9 81.8 80.9 AIME26 96.7 93.3 90.6 93.3 93.3 91.3 General Agent BFCL-V4 63.1 77.5
72.5 67.7 68.3 72.9 TAU2-Bench 87.1 91.6 85.4 84.6 77.0 86.7 VITA-Bench 38.2 56.3 51.6
40.9 41.9 49.7 DeepPlanning 44.6 33.9 23.3 28.7 14.5 34.3 Tool Decathlon 43.8 43.5 36.4
18.8 27.8 38.3 MCP-Mark 57.5 42.3 53.9 33.5 29.5 46.1 Search Agent HLE w/ tool 45.5 43.4
45.8 49.8 50.2 48.3 BrowseComp 65.8 67.8 59.2 53.9 --/74.9 69.0/78.6 BrowseComp-zh 76.1
62.4 66.8 60.9 -- 70.3 WideSearch 76.8 76.4 68.0 57.9 72.7 74.0 Seal-0 45.0 47.7 45.5 46.9
57.4 46.9 Multilingualism MMMLU 89.5 90.1 90.6 84.4 86.0 88.5 MMLU-ProX 83.7 85.7 87.7
78.5 82.3 84.7 NOVA-63 54.6 56.7 56.7 54.2 56.0 59.1 INCLUDE 87.5 86.2 90.5 82.3 83.3
85.6 Global PIQA 90.9 91.6 93.2 86.0 89.3 89.8 PolyMATH 62.5 79.0 81.6 64.7 43.1 73.3
WMT24++ 78.8 79.7 80.7 77.6 77.6 78.9 MAXIFE 88.4 79.2 87.5 84.0 72.8 88.2 Coding Agent
SWE-bench Verified 80.0 80.9 76.2 75.3 76.8 76.4 SWE-bench Multilingual 72.0 77.5 65.0 66.7
73.0 69.3 SecCodeBench 68.7 68.6 62.4 57.5 61.3 68.3 Terminal Bench 2 54.0 59.3 54.2 22.5
50.8 52.5

* HLE-Verified: a verified and revised version of Humanity's Last Exam (HLE), accompanied by a transparent, component-wise verification protocol and a fine-grained error taxonomy. We open-source the dataset at <https://huggingface.co/datasets/skylenage/HLE-Verified>. * TAU2-Bench: we follow the official setup except for the airline

domain, where all models are evaluated by applying the fixes proposed in the Claude Opus 4.5 system card. * MCP-Mark: GitHub MCP server uses v0.30.3 from api.githubcopilot.com; Playwright tool responses are truncated at 32k tokens. * Search Agent: most search agents built on our model adopt a simple context-folding strategy(256k): once the cumulative Tool Response length reaches a preset threshold, earlier Tool Responses are pruned from the history to keep the context within limits. * BrowseComp: we tested two strategies, simple context-folding achieved a score of 69.0, while using the same discard-all strategy as DeepSeek-V3.2 and Kimi K2.5 achieved 78.6. * WideSearch: we use a 256k context window without any context management. * MMLU-ProX: we report the averaged accuracy on 29 languages. * WMT24++: a harder subset of WMT24 after difficulty labeling and rebalancing; we report the averaged scores on 55 languages using XCOMET-XXL. * MAXIFE: we report the accuracy on English + multilingual original prompts (totally 23 settings). * Empty cells (--) indicate scores not yet available or not applicable.

Vision Language

Model	GPT5.2	Claude 4.5 Opus	Gemini-3 Pro	Qwen3-VL-235B-A22B	K2.5-1T-A32B	Qwen3.5-397B-A17B	---
STEM and Puzzle	86.7	80.7	87.2	80.6	84.3	85.0	MMMU-Pro 79.5
MathVision	83.0	74.3	86.6	74.6	84.2	88.6	Mathvista(mini) 83.1 80.0
We-Math	79.0	70.0	86.9	74.8	84.7	87.9	DynaMath 86.8 79.7 85.1 82.8
ZEROBench	9	3	10	4	9	12	ZEROBench_sub 33.2 28.4 39.0 28.4 33.5 41.0
BabyVision	34.4	14.2	49.7	22.2	36.5	52.3/43.3	General VQA RealWorldQA 83.3 77.0 83.3 81.3 81.0 83.9
MMStar	77.1	73.2	83.1	78.7	80.5	83.8	HallusionBench 65.2 64.1 68.6 66.7 69.8 71.4
MMBench EN-DEV-v1.1	88.2	89.2	93.7	89.7	94.2	93.7	SimpleVQA 55.8 65.7 73.2 61.3 71.2 67.1
Text Recognition and Document Understanding							OmniDocBench1.5 85.7 87.7 88.5 84.5 88.8 90.8
CharXiv(RQ)	82.1	68.5	81.4	66.1	77.5	80.8	MMLongBench-Doc -- 61.9 60.5 56.2 58.5 61.5
CC-OCR	70.3	76.9	79.0	81.5	79.7	82.0	AI2D_TEST 92.2 87.7 94.1 89.2 90.8 93.9
OCRBench	80.7	85.8	90.4	87.5	92.3	93.1	Spatial Intelligence ERQA 59.8 46.8 70.5 52.5 -- 67.5
CountBench	91.9	90.6	97.3	93.7	94.1	97.2	RefCOCO(avg) -- -- 84.1 91.1 87.8 92.3 ODInW13 -- -- 46.3 43.2 -- 47.0
EmbSpatialBench	81.3	75.7	61.2	84.3	77.4	84.5	RefSpatialBench -- -- 65.5 69.9 -- 73.6 LingoQA 68.8 78.8 72.8 66.8 68.2 81.6 V* 75.9 67.0 88.0 85.9 77.0 95.8/91.1 Hypersim -- -- -- 11.0 -- 12.5 SUNRGBD -- -- -- 34.9 -- 38.3 Nuscene -- -- -- 13.9 -- 16.0 Video Understanding VideoMME(w sub.) 86 77.6 88.4 83.8 87.4 87.5 VideoMME(w/o sub.) 85.8 81.4 87.7 79.0 83.2 83.7 VideoMMMU 85.9 84.4 87.6 80.0 86.6 84.7 MLVU (M-Avg) 85.6 81.7 83.0 83.8 85.0 86.7 MVBench 78.1 67.2 74.1 75.2 73.5 77.6 LVBench 73.7 57.3 76.2 63.6 75.9 75.5 MMVU 80.8 77.3 77.5 71.1 80.4 75.4 Visual Agent ScreenSpot Pro -- 45.7 72.7 62.0 -- 65.6 OSWorld-Verified 38.2 66.3 -- 38.1 63.3 62.2 AndroidWorld -- -- -- 63.7 -- 66.8 Medical VQA SLAKE 76.9 76.4 81.3 72.5 81.6 79.9 PMC-VQA 58.9 59.9 62.3 56.1 63.3 64.2 MedXpertQA-MM 73.3 63.6 76.0 47.6 65.3 70.0

* MathVision : our model's score is evaluated using a fixed prompt, e.g., "Please reason step by step, and put your final answer within \boxed{}." For other models, we report the higher score between runs with and without the \boxed{} formatting. * BabyVision: our model's score is reported with CI (Code Interpreter) enabled; without CI, the result is 43.3. * V*: our model's score is reported with CI (Code Interpreter) enabled; without CI, the result is 91.1. * Empty cells (--) indicate scores not yet available or not applicable. * Upon review, we found inconsistencies in the evaluation setup of the historical version Qwen3-VL-235B-A22B on SLAKE and PMC-VQA. The corresponding comparative scores were corrected on March 15, 2026.

Compared to the Qwen3 series, the post-training performance gains in Qwen3.5 primarily stem from our extensive scaling of virtually all RL tasks and environments we could conceive. Our approach placed strong emphasis on increasing the difficulty and generalizability of RL environments, rather than optimizing for specific metrics or narrow categories of queries. Below, we illustrate the improvements in general agent capabilities resulting from this RL environment scaling. The overall performance is calculated by averaging the ranking of each model on the following benchmarks: BFCL-V4, VITA-Bench, DeepPlanning, Tool-Decathlon, and MCP-Mark. Additional scaling results across a broader range of tasks will be detailed in our upcoming technical report.

Pretraining

Qwen3.5 advances pretraining across three dimensions—power, efficiency, and versatility:

****Power****: Trained on a significantly larger scale of visual-text tokens compared to Qwen3, with enriched Chinese/

English, multilingual, STEM, and reasoning data under stricter filtering. This enables cross-generation parity: Qwen3.5-397B-A17B matches the >1T-parameter Qwen3-Max-Base.

****Efficiency**:** Built on Qwen3-Next architecture—higher-sparsity MoE, Gated DeltaNet + Gated Attention hybrid attention, stability optimizations, and multi-token prediction. Under the 32k/256k context length, the decoding throughput of Qwen3.5-397B-A17B is 8.6x/19.0x that of Qwen3-Max, and the performance is comparable. The decoding throughput of Qwen3.5-397B-A17B is 3.5x/7.2 times that of Qwen3-235B-A22B.

****Versatility**:** Natively multimodal via early text-vision fusion and expanded visual/STEM/video data, outperforming Qwen3-VL at similar scales. Multilingual coverage grows from 119 to 201 languages/dialects; a 250k vocabulary (vs. 150k) boosts encoding/decoding efficiency by 10–60% across most languages.

Below we present the performance of the base models.

	Qwen3-235B-A22B	GLM-4.5-355B-A32B	DeepSeek-V3.2-671B-A37B	K2-1T-A32B	Qwen3.5-397B-A17B
General Knowledge&Multilingual	87.33	86.56	88.11	87.38	88.61
MMLU-Pro	67.73	65.00	62.82	67.64	76.01
MMLU-Redux	87.44	86.86	87.29	86.65	89.09
SuperGPQA	42.84	44.56	43.46	44.86	57.96
C-Eval	91.82	85.50	90.48	91.82	91.82
MMMLU	81.27	82.26	83.20	82.26	85.82
Include	75.26	73.41	76.52	72.05	79.27
Nova	66.52	60.96	60.40	61.44	67.55
Reasoning&STEM	87.95	87.68	86.03	89.11	90.98
KoRBench	50.80	52.80	54.00	53.84	54.08
GPQA	47.47	44.63	44.16	46.78	54.64
MATH	71.84	61.84	64.40	71.50	74.14
GSM8K	91.17	89.31	89.12	92.12	93.71
Coding	77.60	69.49	62.68	71.77	79.32
Evalplus	77.60	69.49	62.68	71.77	79.32
MultiPLE	65.94	62.51	61.88	70.64	79.39
SWE-agentless	31.77	29.23	34.67	28.54	43.26
CRUX-I	64.25	67.63	63.25	70.50	71.13
CRUX-O	78.88	77.13	73.88	77.13	82.38

Infrastructure

Qwen3.5 enables efficient native multimodal training via a heterogeneous infrastructure that decouples parallelism strategies across vision and language components, avoiding uniform approaches' inefficiencies. By exploiting sparse activations for cross-component computation overlap, it achieves near 100% training throughput versus pure-text baselines on mixed text-image-video data. Complementing this, a native FP8 pipeline applies low precision to activations, MoE routing, and GEMM operations—with runtime monitoring preserving BF16 in sensitive layers—yielding ~50% activation memory reduction and >10% speedup while scaling stably to tens of trillions of tokens.

To continuously unleash the power of reinforcement learning, we built a scalable asynchronous RL framework that supports Qwen3.5 models of all sizes, spanning text, multimodal, and multi-turn settings. By adopting a fully disaggregated training-inference architecture, the framework achieves significantly improved hardware utilization, dynamic load balancing, and fine-grained fault recovery. It further optimizes throughput and enhances train-infer consistency via techniques such as FP8 end-to-end training, rollout router replay, speculative decoding, and multi-turn rollout locking. Through tight system-algorithm co-design, the framework effectively bounds gradient staleness and mitigates data skewness, preserving both training stability and performance. Moreover, it natively supports agentic workflows, facilitating seamless multi-turn interactions without framework-induced interruptions. This decoupled design enables the system to accommodate million-scale agent scaffolds and environments, substantially boosting model generalization. Collectively, these optimizations yield a 3×–5× end-to-end speedup, demonstrating superior stability, efficiency, and scalability.

Play with Qwen3.5

Chat with Qwen3.5

Feel free to use Qwen3.5 on Qwen Chat. We provide three modes, auto, thinking, and fast, to users to choose. With “Auto” mode, users can leverage adaptive thinking, which can think and use tools including search and code interpreter, while with “Thinking” mode, the model can think deeply for hard problems. With “Fast” mode, the model answers questions instantly without spending tokens on thinking.

ModelStudio

Users can experience our flagship model, Qwen3.5-Plus, by invoking it through Alibaba Cloud ModelStudio. To enable advanced capabilities such as reasoning, web search, and Code Interpreter, simply pass the following parameters:

* `enable\_thinking`: Activates reasoning mode (chain-of-thought processing)

* `enable\_search`: Enables web search and Code Interpreter functionality

Example code is provided below:

```
``` """ Environment variables (per official docs): DASHSCOPE_API_KEY: Your API Key from https://
bailian.console.aliyun.com DASHSCOPE_BASE_URL: (optional) Base URL for compatible-mode API.
DASHSCOPE_MODEL: (optional) Model name; override for different models. DASHSCOPE_BASE_URL: -
Beijing: https://dashscope.aliyuncs.com/compatible-mode/v1 - Singapore: https://dashscope-intl.aliyuncs.com/
compatible-mode/v1 - US (Virginia): https://dashscope-us.aliyuncs.com/compatible-mode/v1 """from openai import
OpenAIimport osapi_key = os.environ.get("DASHSCOPE_API_KEY")if not api_key: raise ValueError(
"DASHSCOPE_API_KEY is required. " "Set it via: export DASHSCOPE_API_KEY='your-api-key' ")client =
OpenAI(api_key=api_key, base_url=os.environ.get("DASHSCOPE_BASE_URL", "https://dashscope-
intl.aliyuncs.com/compatible-mode/v1",),)messages = [{"role": "user", "content": "Introduce Qwen3.5."}]model =
os.environ.get("DASHSCOPE_MODEL", "qwen3.5-plus",)completion = client.chat.completions.create(
model=model, messages=messages, extra_body={ "enable_thinking": True, "enable_search": False },
stream=True)reasoning_content = "" # Full reasoning traceanswer_content = "" # Full responseis_answering =
False # Whether we have entered the answer phaseprint("\n" + "=" * 20 + "Reasoning" + "=" * 20 + "\n")for chunk
in completion: if not chunk.choices: print("\nUsage:") print(chunk.usage) continue delta = chunk.choices[0].delta #
Collect reasoning content only if hasattr(delta, "reasoning_content") and delta.reasoning_content is not None: if
not is_answering: print(delta.reasoning_content, end="", flush=True) reasoning_content +=
delta.reasoning_content # Received content, start answer phase if hasattr(delta, "content") and delta.content: if
not is_answering: print("\n" + "=" * 20 + "Answer" + "=" * 20 + "\n") is_answering = True print(delta.content, end="",
flush=True) answer_content += delta.content ```
```

You can effortlessly integrate the Bailian API with third-party coding tools, such as Qwen Code, Claude Code, Cline, OpenClaw, OpenCode, etc., to enable a seamless “vibe coding” experience.

## ## Summary and Future Work

Qwen3.5 provides a strong foundation for universal digital agents through its efficient hybrid architecture and native multimodal reasoning. The next leap requires shifting from model scaling to system integration: building agents with persistent memory for cross-session learning, embodied interfaces for real-world interaction, self-directed improvement mechanisms, and economic awareness to operate within practical constraints. The goal is coherent systems that function autonomously over time, transforming today’s task-bound assistants into persistent, trustworthy partners capable of executing complex, multi-day objectives with human-aligned judgment.

## ## Demo

Now Qwen3.5 playing as an agent is capable of think, search, use tools, and build in the context of multimodality.

Video 15

00:00

/

00:00

## ### Coding & Agents

### #### Web Dev

Qwen3.5 can help with web development, especially for frontend tasks like building web pages and designing user interfaces. It makes creating websites easier by turning simple instructions into working code.

Video 16

00:00

/

00:00

#### #### OpenClaw

Qwen3.5 can be used with OpenClaw to power coding tasks. By integrating with OpenClaw as a third-party agent environment, Qwen3.5 can carry out web search, gather information, and produce structured reports—combining its reasoning and tool use with OpenClaw’s interface for a smooth coding and research experience.

Video 17

00:00

/

00:00

#### #### Qwen Code

With Qwen3.5 as the underlying model, Qwen Code supports “vibe coding”, turning natural-language instructions into code, iterating on projects in real time, and handling creative tasks such as generating videos or other assets. Together, Qwen Code and Qwen3.5 provide a streamlined experience for both everyday programming and exploratory coding.

Video 18

00:00

/

00:00

#### ### Visual Agents

#### #### GUI Agents

Acting as a visual agent for productivity automation, Qwen3.5 enables autonomous interaction with smartphones and computers. As a mobile agent, it can follow natural-language instructions to take actions within mobile apps and enable smooth interaction across multiple apps. As a computer agent, it handles complex, long-horizon desktop workflows, enabling office automation.

Fill the missing rows and columns which show the total value

Qwen3.5

Video 19

00:00

/

00:00

#### #### Visual Coding

With its input length expanded to one million tokens, Qwen3.5 can process up to two hours of video. This unlocks a range of powerful applications, from turning hand-drawn UI sketches into clean frontend code, to reverse-engineering logic from simple gameplay footage, to condensing long videos into structured web pages and visual summaries.

复刻这个小游戏的 HTML 代码

Video 20

00:00

Qwen3.5

Video 21

00:00

/

00:00

#### #### Spatial Intelligence

By modeling pixel-level spatial relationships in images, Qwen3.5 achieves higher accuracy in tasks like object counting, relative positioning, and spatial description. It effectively mitigates errors caused by perspective shifts or occlusions—showcasing strong potential for embodied AI applications such as autonomous driving scene understanding and robotic navigation.

What you are seeing are frames from a video sequence. Your answer should focus on the final dynamic outcome, rather than analyzing the results of each individual frame. Why didn't you stop at this intersection?

Qwen3.5

<think>

The user wants to know why the vehicle didn't stop at the intersection based on the provided video frames.

1. **Analyze the sequence of events:**

**Frames 1-3:** The vehicle is approaching an intersection. The traffic lights ahead are visible. In Frame 1 and 2, the lights are green. In Frame 3, the lights are still green. **Frame 4:** The vehicle is closer to the stop line (marked with a bicycle symbol). The traffic lights have turned amber (yellow). The vehicle is already very close to the intersection. **Frame 5:** The vehicle has crossed the stop line and is entering the intersection. The traffic lights are amber. The vehicle is proceeding through.

2. **Determine the traffic rule:** In the UK (driving on the left, red double-decker bus), an amber light means "Stop at the stop line." However, there is a crucial exception: "You may go on only if the AMBER appears after you have crossed the stop line or are so close to it that to pull up might cause an accident."

3. **Apply the rule to the visual evidence:**

In Frame 3, the light is green, and the car is approaching. In Frame 4, the light turns amber. At this exact moment, the car is extremely close to the white stop line (the box with the bicycle symbol). Stopping abruptly from that distance and speed would likely be dangerous or cause a rear-end collision (hard braking). The car is already committed to the intersection. Therefore, the driver continued through because they were too close to stop safely when the light changed from green to amber.

4. **Formulate the answer:**

Start by identifying the state of the traffic lights. They were green as the car approached. Note the change. As the car got very close to the intersection (specifically the advanced stop line for cyclists), the lights turned amber. Explain the reasoning. Because the vehicle was already so close to the stop line when the light changed, stopping safely would have been difficult or dangerous. It's the standard "point of no return" scenario. The driver proceeded through the amber light legally and safely because they were too close to stop.

5. **Refine the answer:** Ensure it directly answers "Why didn't you stop?"

"I didn't stop because the traffic lights turned amber just as I was approaching the stop line." "I was too close to the intersection to stop safely." "The lights were green initially, allowing me to approach."

Let's combine these points. The primary reason is proximity to the stop line when the signal changed.

**Self-Correction/Check:** Did I miss anything? Frame 1: Green light. Frame 2: Green light. Frame 3: Green light. Frame 4: Amber light. The car is right at the bicycle box. Frame 5: Amber light. The car is past the box.

Conclusion stands: The light changed to amber when the vehicle was too close to stop safely.

</think>

I did not stop because the traffic lights turned amber just as I was approaching the stop line.

Looking at the sequence:

1. In the earlier frames, the traffic lights are clearly green, so I continued driving towards the intersection. 2. By the

fourth frame, as I reached the advanced stop line (the box with the bicycle symbol), the lights changed to amber. 3. At that specific moment, I was too close to the intersection to stop safely without slamming on the brakes, so I proceeded through the junction while the light was amber.

Is the telephone booth on the left or right side of the yellow van?

Qwen3.5

<think>

The user wants to know the relative position of the telephone booth compared to the yellow van.

1. **\*\*Locate the yellow van:\*\*** I need to scan the street in the lower middle part of the image. I see a line of parked cars and moving vehicles. There is a distinct yellow van [403, 776, 423, 804] parked on the right side of the road (from the viewer's perspective, it's driving away or parked facing away, but actually, looking closely, it's facing towards the camera, wait no, let's re-evaluate). Let's look at the traffic flow. The cars on the left are facing towards the camera (headlights visible? hard to tell, but they are parked). The cars on the right seem to be parked or moving. Wait, let's look at the yellow vehicle. It's a yellow van [403, 776, 423, 804]. 2. **\*\*Locate the telephone booth:\*\*** I need to find a red telephone booth. Scanning near the yellow van. Just to the right of the yellow van, further down the sidewalk/street area, there is a red object. Let's zoom in mentally. Ah, there is a red telephone booth [449, 796, 461, 831] standing on the sidewalk. 3. **\*\*Determine relative position:\*\*** \* The yellow van is at roughly  $x=403$  to  $x=423$ . \* The telephone booth is at roughly  $x=449$  to  $x=461$ . \* Since the x-coordinate of the telephone booth is greater than the x-coordinate of the yellow van, the telephone booth is to the right of the yellow van from the viewer's perspective.

Let me double-check.

\* Yellow van: [403, 776, 423, 804] \* Telephone booth: [449, 796, 461, 831] \* The booth is clearly to the right of the van in the image frame.

Wait, is that definitely a telephone booth? Yes, it's a classic red British phone box. Is that definitely a yellow van? Yes, it looks like a small yellow delivery van or similar vehicle.

So, the telephone booth is on the right side of the yellow van.

</think>

The telephone booth is on the right side of the yellow van.

Qwen3.5

#### Thinking with Images

Qwen3.5 can use tools such as a code interpreter and image search during multimodal reasoning. This means it can not only interpret images, but also leverage external tools during its reasoning — transforming images, rendering intermediate reasoning steps, and using visuals to verify its textual results.

画一画，找出红球到绿球的最短路线

(En: Draw it out and find the shortest path from the red ball to the green ball.)

Qwen3.5

<think>

为了找到从红球到绿球的最短路线，我首先需要将图像转换为一个网格表示，识别出起点（红球）、终点（绿球）以及墙壁和路径。然后，我将使用广度优先搜索（BFS）算法来找到最短路径。

首先，我将读取图像，将其转换为灰度并二值化，以区分墙壁和路径。然后，我将检测红球和绿球的位置。

</think>

```
`import cv2import numpy as npimport matplotlib.pyplot as pltimport collections# Load the imageimage = np.array(image_clue[0])# The maze seems to be a grid. Let's try to determine the grid size.# The image has a thick black border.# Let's convert to HSV to detect red and green colors for start and end points.hsv = cv2.cvtColor(image, cv2.COLOR_RGB2HSV)# Define range of red colorslower_red = np.array([0, 70,
```

```

50))upper_red = np.array([10, 255, 255])# Threshold the HSV image to get red componentsmask1 =
cv2.inRange(hsv, lower_red, upper_red)# Sometimes red wraps around 180, so check upper end too
lower_red2 = np.array([160, 70, 50])upper_red2 = np.array([180, 255, 255])mask2 = cv2.inRange(hsv, lower_red2,
upper_red2)mask_red = cv2.bitwise_or(mask1, mask2)# Define range of green colors
lower_green = np.array([40, 70, 50])upper_green = np.array([80, 255, 255])mask_green = cv2.inRange(hsv, lower_green, upper_green)# Find
coordinates of red and green pixels
red_pixels = np.column_stack(np.where(mask_red > 0))green_pixels = np.column_stack(np.where(mask_green > 0))
print(f"Red pixels count: {len(red_pixels)}")print(f"Green pixels count: {len(green_pixels)}")# Let's estimate grid size.# The image looks like a square maze. Let's assume a standard
size like 15x15 or similar.h, w = image.shape[:2]print(f"Image dimensions: {w}x{h}")# Let's try to detect the grid
lines or infer from the structure.# The walls are black, paths are white.gray = cv2.cvtColor(image,
cv2.COLOR_RGB2GRAY)# Threshold to separate black walls from white paths_, binary = cv2.threshold(gray,
128, 255, cv2.THRESH_BINARY)# Let's try to infer grid size by looking at runs of pixels.# Or simply assume a
size based on visual inspection.# Let's count transitions or look at the structure.# Let's try to resize the image to
various grid sizes and see which one makes sense.# Common sizes are 10x10, 15x15, 20x20, etc.# Let's try to
detect the bounding box of the maze content first.coords = cv2.findNonZero(255 - binary) # Find black pixels
(walls)x, y, w_maze, h_maze = cv2.boundingRect(coords)print(f"Maze bounding box: x={x}, y={y}, w={w_maze},
h={h_maze}")# It seems the maze fills most of the image.# Let's try to determine grid size by checking common
divisors or visual estimation.# Let's assume the maze is roughly square.# Let's try to print out potential grid sizes.#
If I look at the image, it seems to be around 17x17 or similar.# Let's try to detect lines.# Let's try a simpler
approach: assume the maze is a grid of N x N cells.# Let's try to find N such that the cell size is consistent.# Let's
try N from 10 to 30.# The bounding box is likely the whole image or close to it.# Let's assume the black border is
part of the maze walls.# Let's try to infer grid size by looking at the image dimensions.# 512x512 is common, or
similar. Let's check w, h again.# Let's try to divide w and h by integers to see if we get a reasonable cell size.#
Let's guess the grid size is roughly 17x17 based on visual complexity.# Let's try to detect the grid by looking at the
binary image.# Let's try to simply assume a grid size and see if it reconstructs well.# Let's try to count the number
of black/white transitions in the middle row/col.# Let's try to detect the grid size by checking the run lengths in the
binary image.def estimate_grid_size(binary_img): # Scan middle row and column
mid_row = binary_img[h//2, :]
mid_col = binary_img[:, w//2] # This might be noisy. Let's try to just guess based on image size. # Let's print image
size first.
returnprint(f"Image shape: {image.shape}")# Let's try to visualize the masks to confirm start and end
points.plt.figure(figsize=(15, 5))plt.subplot(1, 3, 1)plt.imshow(image)plt.title(f"Original Image ({w}
x{h})")plt.axis('off')plt.subplot(1, 3, 2)plt.imshow(mask_red, cmap='gray')plt.title(f"Red Mask (Start):
{len(red_pixels)} pixels")plt.axis('off')plt.subplot(1, 3, 3)plt.imshow(mask_green, cmap='gray')plt.title(f"Green Mask
(End): {len(green_pixels)} pixels")plt.axis('off')plt.tight_layout()plt.show()# Calculate centroids for start and end
if len(red_pixels) > 0: start_y, start_x = np.mean(red_pixels, axis=0).astype(int) print(f"Start point (pixel): ({start_x},
{start_y})")else: start_x, start_y = -1, -1if len(green_pixels) > 0: end_y, end_x = np.mean(green_pixels,
axis=0).astype(int) print(f"End point (pixel): ({end_x}, {end_y})")else: end_x, end_y = -1, -1

```

Red pixels count: 279 Green pixels count: 238 Image dimensions: 612x606 Maze bounding box: x=15, y=15, w=587, h=586 Image shape: (606, 612, 3) Start point (pixel): (61, 60) End point (pixel): (555, 554)

Start%20point%20(pixel):%20(61,%2060)End%20point%20(pixel):%20(555,%20554))

Qwen3.5

<think>

图像已成功加载并处理。我检测到了红球和绿球的位置，以及迷宫的边界框。

\* 图像尺寸：612x606 \* 迷宫边界框：x=15, y=15, w=587, h=586 \* 红球中心（像素坐标）：(61, 60) \* 绿球中心（像素坐标）：(555, 554)

接下来，我需要确定迷宫的网格大小。从边界框的宽度和高度（587x586）来看，这似乎是一个正方形区域。通常迷宫是由相同大小的单元格组成的。让我尝试推断网格的尺寸。假设迷宫是一个 N x N 的网格。587 和 586 非常接近。如果这是一个 17x17 的网格（包括墙壁），每个单元格大约是 34-35 像素。让我尝试通过检查图像中的线条或简单的除法来确定网格大小。587 / 17 = 34.52... 586 / 17 = 34.47... 这看起来不太像整数。让我们尝试其他常见的尺寸，比如 15x15, 19x19, 21x21 等。或者，我可以尝试根据红球和绿球的位置以及边界框来推断。红球在左上角，绿球在右下角。让我尝试将边界框内的图像分割成网格，并尝试几种可能的网格尺寸，看看哪种能产生清晰的黑白模式。考虑到边界框是 587x586，这可能意味着最外圈是墙壁。让我们尝试检测边缘或简单的阈值来估算单元格大小。或者，我可以假设网格大小是使得 587 和 586 能被整除的数。587 是质数吗？587 / 1 = 587. 586

= 2 \* 293. 这有点奇怪。也许边界框检测有轻微偏差，或者迷宫本身不是完美的正方形网格，或者最外层的黑边厚度不同。让我重新检查边界框。也许我应该直接对整个图像进行网格化，或者尝试找到一个更规则的子区域。让我们尝试打印出边界框内图像的一些统计信息，或者尝试假设一个网格大小并可视化它。常见的迷宫尺寸如 10x10, 15x15, 20x20 等。如果单元格大小是 34 像素，17 \* 34 = 578。如果单元格大小是 35 像素，17 \* 35 = 595。如果单元格大小是 30 像素，19 \* 30 = 570，20 \* 30 = 600。让我们尝试自动检测网格大小。我可以计算水平和垂直方向上的投影，或者寻找重复的模式。另一种方法是，假设迷宫由黑色墙壁和白色路径组成。让我尝试将边界框内的图像调整大小到不同的网格尺寸，看看哪个看起来最清晰。但首先，让我确认边界框是否准确。从显示的裁剪图来看，边界框似乎包含了整个迷宫区域。让我们尝试假设网格大小为 17x17 或 19x19 或类似，并查看平均颜色。或者，我可以简单地尝试将图像划分为 N x N 个单元格，其中 N 使得单元格大小接近整数。587 和 586 非常接近 588 (which is 17\_34.58... no, 17\_34=578, 17\_35=595). 也许网格是 15x15? 587/15 = 39.13. 也许是 20x20? 587/20 = 29.35. 也许是 21x21? 587/21 = 27.95. 接近 28. 21\_28 = 588. 如果是 21x21 的网格，每个单元格 28 像素，总宽 588。边界框宽 587，非常接近。如果是 21x21 的网格，每个单元格 28 像素，总高 588。边界框高 586，也非常接近。这很有希望。让我尝试用 21x21 的网格来解析迷宫。单元格大小约为 28 像素。起始点 (61, 60) 相对于边界框 (15, 15) 是 (46, 45)。46/28 = 1.64, 45/28 = 1.60. 这看起来像是在 (1,1) 或 (2,2) 附近。等等，红球是在左上角的白色路径里。通常迷宫的入口/出口在边缘。看原图，红球在左上角的一个白色格子里，周围有黑边。让我们尝试用代码来确认网格大小和解析迷宫。我将尝试几种可能的网格尺寸（例如 17x17, 19x19, 21x21），并检查哪种尺寸产生的二值化网格最一致（即每个单元格要么是纯黑，要么是纯白，或者有明显的中心点颜色）。

修正思路：

1. 提取边界框内的图像。2. 尝试将图像划分为 N x N 网格。3. 对于每个网格单元，计算平均颜色。4. 根据颜色判断是墙壁还是路径。5. 确定起点和终点的网格坐标。6. 使用 BFS 寻找最短路径。7. 绘制路径。

我将首先尝试推断网格大小。边界框宽 587，高 586。可能的网格尺寸 N 和单元格大小 S：N=17, S=34.5 N=19, S=30.9 N=21, S=27.95 -> 28 N=29, S=20.2 N=31, S=18.9 让我尝试 N=21, S=28。总尺寸 588x588。边界框 587x586。误差很小，可能是裁剪或抗锯齿造成的。让我尝试 N=17, S=34.5 (不太可能)。让我尝试 N=15, S=39.1。让我尝试 N=29, S=20. 让我尝试 N=30, S=19.5。

我将编写代码来尝试 N=21 (S=28) 和 N=17 (S=34) 等，并可视化网格以确认。同时，我也需要处理起点和终点。红球和绿球可能占据整个单元格或部分单元格。从掩膜图看，红球和绿球是明显的圆点。我将使用之前检测到的红球和绿球中心坐标来定位它们在网格中的位置。相对坐标：Start: (61-15, 60-15) = (46, 45) End: (555-15, 554-15) = (540, 539)

如果 S=28: Start cell: (46/28, 45/28) = (1.64, 1.60) -> 可能是 (1,1) 或 (2,2) 索引？End cell: (540/28, 539/28) = (19.28, 19.25) -> 可能是 (19,19) 或 (20,20)？如果网格是 21x21，索引 0-20。(1,1) 和 (19,19) 看起来是合理的内部位置，或者靠近边缘。看原图，红球在左上角第二个格子（如果第一个是黑边）？或者就是角落的格子。通常迷宫最外圈是墙壁。如果最外圈是墙壁，那么路径从 (1,1) 开始。让我们尝试用代码来验证。

</think>

```
`import cv2import numpy as npimport matplotlib.pyplot as pltimport collections# Load the imageimage = np.array(image_clue[0])# The maze seems to be a grid. Let's try to determine the grid size.# The image has a thick black border.# Let's convert to HSV to detect red and green colors for start and end points.hsv = cv2.cvtColor(image, cv2.COLOR_RGB2HSV)# Define range of red colorslower_red = np.array([0, 70, 50])upper_red = np.array([10, 255, 255])# Threshold the HSV image to get red componentsmask1 = cv2.inRange(hsv, lower_red, upper_red)# Sometimes red wraps around 180, so check upper end toolower_red2 = np.array([160, 70, 50])upper_red2 = np.array([180, 255, 255])mask2 = cv2.inRange(hsv, lower_red2, upper_red2)mask_red = cv2.bitwise_or(mask1, mask2)# Define range of green colorslower_green = np.array([40, 70, 50])upper_green = np.array([80, 255, 255])mask_green = cv2.inRange(hsv, lower_green, upper_green)# Find coordinates of red and green pixelsred_pixels = np.column_stack(np.where(mask_red > 0))green_pixels = np.column_stack(np.where(mask_green > 0))print(f"Red pixels count: {len(red_pixels)}")print(f"Green pixels count: {len(green_pixels)}")# Let's estimate grid size.# The image looks like a square maze. Let's assume a standard size like 15x15 or similar.h, w = image.shape[:2]print(f"Image dimensions: {w}x{h}")# Let's try to detect the grid lines or infer from the structure.# The walls are black, paths are white.gray = cv2.cvtColor(image, cv2.COLOR_RGB2GRAY)# Threshold to separate black walls from white paths_, binary = cv2.threshold(gray, 128, 255, cv2.THRESH_BINARY)# Let's try to infer grid size by looking at runs of pixels.# Or simply assume a size based on visual inspection.# Let's count transitions or look at the structure.# Let's try to resize the image to various grid sizes and see which one makes sense.# Common sizes are 10x10, 15x15, 20x20, etc.# Let's try to
```

```

detect the bounding box of the maze content first.coords = cv2.findNonZero(255 - binary) # Find black pixels
(walls)x, y, w_maze, h_maze = cv2.boundingRect(coords)print(f"Maze bounding box: x={x}, y={y}, w={w_maze},
h={h_maze}")# It seems the maze fills most of the image.# Let's try to determine grid size by checking common
divisors or visual estimation.# Let's assume the maze is roughly square.# Let's try to print out potential grid sizes.#
If I look at the image, it seems to be around 17x17 or similar.# Let's try to detect lines.# Let's try a simpler
approach: assume the maze is a grid of N x N cells.# Let's try to find N such that the cell size is consistent.# Let's
try N from 10 to 30.# The bounding box is likely the whole image or close to it.# Let's assume the black border is
part of the maze walls.# Let's try to infer grid size by looking at the image dimensions.# 512x512 is common, or
similar. Let's check w, h again.# Let's try to divide w and h by integers to see if we get a reasonable cell size.#
Let's guess the grid size is roughly 17x17 based on visual complexity.# Let's try to detect the grid by looking at the
binary image.# Let's try to simply assume a grid size and see if it reconstructs well.# Let's try to count the number
of black/white transitions in the middle row/col.# Let's try to detect the grid size by checking the run lengths in the
binary image.def estimate_grid_size(binary_img): # Scan middle row and column mid_row = binary_img[h//2, :]
mid_col = binary_img[:, w//2] # This might be noisy. Let's try to just guess based on image size. # Let's print image
size first. returnprint(f"Image shape: {image.shape}")# Let's try to visualize the masks to confirm start and end
points.plt.figure(figsize=(15, 5))plt.subplot(1, 3, 1)plt.imshow(image)plt.title(f"Original Image ({w}
x{h})")plt.axis('off')plt.subplot(1, 3, 2)plt.imshow(mask_red, cmap='gray')plt.title(f"Red Mask (Start):
{len(red_pixels)} pixels")plt.axis('off')plt.subplot(1, 3, 3)plt.imshow(mask_green, cmap='gray')plt.title(f"Green Mask
(End): {len(green_pixels)} pixels")plt.axis('off')plt.tight_layout()plt.show()# Calculate centroids for start and end if
len(red_pixels) > 0: start_y, start_x = np.mean(red_pixels, axis=0).astype(int) print(f"Start point (pixel): ({start_x},
{start_y})")else: start_x, start_y = -1, -1if len(green_pixels) > 0: end_y, end_x = np.mean(green_pixels,
axis=0).astype(int) print(f"End point (pixel): ({end_x}, {end_y})")else: end_x, end_y = -1, -1`

```

```

Grid size: 21x21, Cell size: 28 Cluster centers: [[47.1935085] [200.60667539]] Start pixel (relative): (46, 45) ->
Grid: (1, 1) End pixel (relative): (540, 539) -> Grid: (19, 19) Maze grid shape: (21, 21) Start cell value: 1, End cell
value: 1 Path found: [(1, 1), (1, 2), (1, 3), (2, 3), (3, 3), (4, 3), (5, 3), (6, 3), (7, 3), (8, 3), (8, 4), (8, 5), (8, 6), (8, 7),
(8, 8), (7, 8), (6, 8), (5, 8), (4, 8), (3, 8), (2, 8), (1, 8), (1, 9), (1, 10), (1, 11), (1, 12), (2, 12), (3, 12), (3, 13), (3, 14),
(4, 14), (5, 14), (6, 14), (6, 13), (6, 12), (6, 11), (6, 10), (7, 10), (8, 10), (8, 11), (8, 12), (9, 12), (10, 12), (10, 11),
(10, 10), (10, 9), (10, 8), (10, 7), (10, 6), (11, 6), (12, 6), (12, 5), (12, 4), (12, 3), (11, 3), (10, 3), (10, 2), (10, 1), (11,
1), (12, 1), (13, 1), (14, 1), (15, 1), (16, 1), (17, 1), (17, 2), (17, 3), (18, 3), (19, 3), (19, 4), (19, 5), (19, 6), (19, 7),
(19, 8), (18, 8), (17, 8), (17, 9), (16, 9), (16, 10), (15, 10), (14, 10), (13, 10), (12, 10), (12, 11), (12, 12), (13, 12),
(14, 12), (15, 12), (16, 12), (17, 12), (17, 13), (17, 14), (18, 14), (19, 14), (19, 15), (19, 16), (19, 17), (19, 18), (19,
19)] Start point (pixel): (61, 60) End point (pixel): (555, 554)

```

```

Start%20point%20(pixel):%20(61,%2060)End%20point%20(pixel):%20(555,%20554))

```

Qwen3.5

<think>

代码成功解析了迷宫并找到了路径。

\* 网格大小：21x21 \* 单元格大小：28 \* 起点网格坐标：(1, 1) \* 终点网格坐标：(19, 19) \* 路径长度：99 步 \* 右侧图像显示了找到的路径（红色），从绿色起点到蓝色终点。

路径看起来是连贯的，并且遵循白色路径。我现在需要生成最终的图像，将路径绘制在原始图像上，或者像右侧那样显示解决方案。用户要求“画一画，找出红球到绿球的最短路线”。我将创建一个图像，在原始迷宫上叠加红色的路径线。为了更美观，我可以将路径坐标转换回像素坐标并绘制线条或填充单元格。由于我已经有了网格化的表示，我可以直接在调整大小后的图像上绘制，或者映射回原始坐标。考虑到原始图像有边界框，我将把路径映射回原始图像的坐标系。

映射逻辑：网格坐标 (r, c) 对应于边界框内的像素区域。边界框左上角：(15, 15) 单元格大小：28 网格单元 (r, c) 的中心像素坐标（相对于边界框）：(c\_28 + 14, r\_28 + 14) 加上边界框偏移：(15 + c\_28 + 14, 15 + r\_28 + 14) = (c\_28 + 29, r\_28 + 29) 或者简单地绘制填充矩形。

让我在原始图像上绘制路径。我将使用 OpenCV 在原始图像副本上绘制红色线条或矩形。为了清晰，我将绘制连接路径中每个单元格中心的线条，线宽设为单元格大小的一半或更小。或者，我可以简单地填充路径经过的单元格。让我们尝试绘制连接中心的线条，颜色为红色，线宽为 10 像素左右。

首先，我需要重新加载原始图像（或者使用之前加载的），然后应用路径绘制。由于我无法直接访问之前的变量，我需要在新的代码块中重新定义必要的变量或重新运行部分代码。不过，我可以简单地使用之前计算出的路径坐标

和参数。路径坐标：`path` 变量。边界框：(15, 15) 单元格大小：28

我将编写代码来：

1. 加载原始图像。 2. 定义路径坐标列表。 3. 将路径坐标转换为像素坐标。 4. 在图像上绘制路径。 5. 显示结果。

</think>

```
`import cv2import numpy as npimport matplotlib.pyplot as plt# Load the original imageimage_path =
image_clue[0]image = np.array(image_path)# Parameters from previous stepbbox_x, bbox_y = 15, 15cell_size =
28# Path coordinates from previous executionpath_coords = [(1, 1), (1, 2), (1, 3), (2, 3), (3, 3), (4, 3), (5, 3), (6, 3),
(7, 3), (8, 3), (8, 4), (8, 5), (8, 6), (8, 7), (8, 8), (7, 8), (6, 8), (5, 8), (4, 8), (3, 8), (2, 8), (1, 8), (1, 9), (1, 10), (1, 11),
(1, 12), (2, 12), (3, 12), (3, 13), (3, 14), (4, 14), (5, 14), (6, 14), (6, 13), (6, 12), (6, 11), (6, 10), (7, 10), (8, 10), (8,
11), (8, 12), (9, 12), (10, 12), (10, 11), (10, 10), (10, 9), (10, 8), (10, 7), (10, 6), (11, 6), (12, 6), (12, 5), (12, 4), (12,
3), (11, 3), (10, 3), (10, 2), (10, 1), (11, 1), (12, 1), (13, 1), (14, 1), (15, 1), (16, 1), (17, 1), (17, 2), (17, 3), (18, 3),
(19, 3), (19, 4), (19, 5), (19, 6), (19, 7), (19, 8), (18, 8), (17, 8), (17, 9), (16, 9), (16, 10), (15, 10), (14, 10), (13, 10),
(12, 10), (12, 11), (12, 12), (13, 12), (14, 12), (15, 12), (16, 12), (17, 12), (17, 13), (17, 14), (18, 14), (19, 14), (19,
15), (19, 16), (19, 17), (19, 18), (19, 19)]# Create a copy of the image to draw onimage_copy = image.copy()#
Function to get pixel center from grid coordinatedef get_pixel_center(r, c): x = bbox_x + c * cell_size + cell_size //
2 y = bbox_y + r * cell_size + cell_size // 2 return (x, y)# Draw the path# We can draw lines between centers or fill
rectangles. Filling rectangles might look blocky but accurate.# Drawing lines with a thick width is usually
smoother.points = [get_pixel_center(r, c) for r, c in path_coords]points_np = np.array(points,
dtype=np.int32).reshape((-1, 1, 2))# Draw lines# Color: Red (255, 0, 0)# Thickness: 10cv2.polylines(image_copy,
[points_np], isClosed=False, color=(255, 0, 0), thickness=12)# Also mark start and end points clearly# Start:
Green (0, 255, 0) - wait, original start is red, end is green.# Let's keep original colors or use distinct ones. The
user asked to find the route from red to green.# I'll draw the path in a distinct color, maybe blue or keep red but
thicker.# Let's use a bright magenta or just red with high thickness.# Let's re-draw with a slightly different color to
stand out, e.g., (0, 255, 255) Cyan or (255, 0, 255) Magenta.# But red is fine if it's thick enough. Let's use (255,
50, 50) to be slightly different from the start dot.# Actually, let's just use the red color (255, 0, 0) as requested "red
ball to green ball".# Let's re-draw with thickness 14 to cover the path well.cv2.polylines(image_copy, [points_np],
isClosed=False, color=(255, 0, 0), thickness=14)# Highlight start and endstart_pt = get_pixel_center(1, 1)end_pt =
get_pixel_center(19, 19)cv2.circle(image_copy, start_pt, 8, (255, 0, 0), -1) # Red startcv2.circle(image_copy,
end_pt, 8, (0, 128, 0), -1) # Green endplt.figure(figsize=(15,
15))plt.imshow(image_copy)plt.axis('off')plt.tight_layout()plt.show()`
```

Qwen3.5

<think>

代码成功执行，并在原始图像上绘制了从红球到绿球的最短路径（红色线条）。路径看起来正确地沿着白色通道行进，避开了黑色墙壁。起点和终点也正确对应。现在我可以向用户展示结果并简要说明。

</think>

这是从红球到绿球的最短路线图：

**\*\*解题步骤\*\*：**

1. **\*\*图像识别\*\***：首先识别出迷宫的边界、墙壁（黑色区域）和路径（白色区域），并定位红球（起点）和绿球（终点）的位置。 2. **\*\*网格化\*\***：将迷宫划分为 21x21 的网格，每个单元格大小为 28x28 像素。 3. **\*\*路径搜索\*\***：使用广度优先搜索（BFS）算法在网格中寻找从起点到终点的最短路径。 4. **\*\*绘制结果\*\***：将找到的路径坐标映射回原始图像像素坐标，并绘制红色线条。

红色线条即为最短路线，共经过 99 个步骤。

#### Visual Reasoning

Compared to Qwen3-VL, Qwen3.5 exhibits more robust performance in scientific problem-solving and other visual reasoning tasks. By integrating image content with contextual understanding, it can carry out multi-step logical reasoning, providing a more dependable foundation for multimodal agents in education, research, and beyond.

Hint: Please answer the question and provide the final answer at the end. Question: All vehicles in the garage can only drive forwards or backwards. The black car wants to leave the garage (see diagram). What is the minimum

number of grey vehicles that need to move at least a little bit so that this is possible?

Qwen3.5

<think>

The user wants me to solve a parking lot puzzle.

**\*\*1. Analyze the image:\*\***

\* It's a grid-based puzzle, likely 6x6. \* There's a black car that needs to exit to the right. \* The exit is on the right side, in the third row from the top. \* There are several grey vehicles blocking the way.

**\*\*2. Identify the vehicles and their positions:\*\*** Let's use a coordinate system (row, column) from top-left (1,1) to bottom-right (6,6).

\* **\*\*Black Car:\*\*** Horizontal, located at (3, 2) and (3, 3). Wait, let's look closer. \* Row 1: A small grey car horizontal at (1, 2)-(1, 3). \* Row 2: A long grey truck/bus vertical at (2, 1)-(4, 1). Wait, let's re-examine. \* Let's just look at the grid. It looks like a 6x6 grid. \* **\*\*Exit:\*\*** Row 3, exiting to the right. \* **\*\*Target:\*\*** The black car is in Row 3. It's a horizontal car. It occupies roughly columns 2 and 3. Wait, let me look really closely at the black car. It's in the second row from the top? No. \* Let's count grid squares. \* Top row: Small grey car (horizontal) at col 2-3. \* Second row down: \* Left edge: Vertical grey truck/bus. It spans rows 2, 3, 4 in column 1. \* Middle: Vertical grey bus. It spans rows 2, 3, 4 in column 4. \* Right edge: Vertical grey car. It spans rows 1, 2 in column 6. Wait, let's look at the top right car. It's vertical. It's in column 6. It seems to span rows 1 and 2.

\* Third row down: \* **\*\*Black car:\*\*** Horizontal. It's in row 3. It spans columns 2 and 3. Wait, let's look at the grid lines. \* Let's assume standard Rush Hour grid (6x6). \* **\*\*Row 1:\*\*** \* Col 2-3: Small grey car (horizontal). \* Col 6: Small grey car (vertical), top part in row 1.

\* **\*\*Row 2:\*\*** \* Col 1: Long grey truck (vertical). Top is in row 2. \* Col 4: Long grey bus (vertical). Top is in row 2. \* Col 6: Small grey car (vertical). Bottom part is in row 2.

\* **\*\*Row 3:\*\*** \* Col 1: Long grey truck (vertical). Middle part. \* **\*\*Black car:\*\*** Horizontal. It is in row 3. It occupies col 2 and 3. \* Col 4: Long grey bus (vertical). Middle part. \* Exit is to the right of row 3.

\* **\*\*Row 4:\*\*** \* Col 1: Long grey truck (vertical). Bottom part. \* Col 4: Long grey bus (vertical). Bottom part.

\* **\*\*Row 5:\*\*** \* Col 1: Small grey car (vertical). Top part. \* Col 5-6: Small grey car (horizontal).

\* **\*\*Row 6:\*\*** \* Col 1: Small grey car (vertical). Bottom part. \* Col 3-5: Long grey bus (horizontal).

Let's refine the positions based on visual inspection.

\* **\*\*Grid:\*\*** 6x6. \* **\*\*Exit:\*\*** Row 3, right side. \* **\*\*Black Car:\*\*** Horizontal, Row 3, Cols 2-3. Wait, looking at the black car, it's in the second row of "vehicle slots"? No, let's count grid cells. \* Top-left cell is empty? No, there's a vertical truck on the left. \* Let's count cells horizontally. 1, 2, 3, 4, 5, 6. \* Let's count cells vertically. 1, 2, 3, 4, 5, 6. \* **\*\*Vehicle 1 (Top Left-ish):\*\*** Horizontal grey car. Row 1, Cols 2-3. \* **\*\*Vehicle 2 (Left Edge):\*\*** Vertical grey truck. Cols 1, Rows 2-4. (Length 3). \* **\*\*Vehicle 3 (Top Right):\*\*** Vertical grey car. Col 6, Rows 1-2. (Length 2). \* **\*\*Vehicle 4 (Middle):\*\*** Vertical grey bus. Col 4, Rows 2-4. (Length 3). \* **\*\*Vehicle 5 (The Target):\*\*** Black car. Horizontal. Row 3, Cols 2-3. Wait, looking at the grid lines... \* The black car is in the row below the top horizontal car. That's Row 2? No. \* Let's look at the vertical truck on the left. It spans 3 grid units. It starts at row 2. So it occupies (2,1), (3,1), (4,1). \* The black car is to the right of it. So it's in Row 3. It occupies (3,2) and (3,3). \* Wait, is the black car length 2 or 3? It looks like a standard car, length 2. So (3,2) and (3,3).

\* **\*\*Vehicle 6 (Bottom Left):\*\*** Vertical grey car. Col 1, Rows 5-6. (Length 2). \* **\*\*Vehicle 7 (Bottom Middle):\*\*** Horizontal grey bus. Row 6, Cols 3-5. (Length 3). Wait, let's look at the bottom bus. It spans cols 3, 4, 5, 6? No, looks like cols 3, 4, 5. Let's assume length 3. \* **\*\*Vehicle 8 (Bottom Right-ish):\*\*** Horizontal grey car. Row 5, Cols 5-6. Wait, looking at the car above the bottom bus. It's in Row 5. It occupies Cols 5-6. No, looking at the grid, it's in Row 5, Cols 5-6? Or Row 5, Cols 4-5? \* Let's re-evaluate the grid alignment. \* The vertical bus in the middle is in Col 4. \* The car to its right (bottom right area) is in Row 5. It looks like it's in Cols 5-6. Wait, let's look at the car above the bottom bus. \* Let's trace the grid lines carefully. \* **\*\*Row 1:\*\*** \* (1,1): Empty. \* (1,2)-(1,3): Horizontal grey car. \* (1,4): Empty. \* (1,5): Empty. \* (1,6): Top part of vertical car.

\* **\*\*Row 2:\*\*** \* (2,1): Top part of vertical truck. \* (2,2): Empty. \* (2,3): Empty. \* (2,4): Top part of vertical bus. \* (2,5):

Empty. \* (2,6): Bottom part of vertical car.

\* \*\*Row 3:\*\* \* (3,1): Middle part of vertical truck. \* (3,2)-(3,3): \*\*Black Car\*\*. \* (3,4): Middle part of vertical bus. \* (3,5): Empty. \* (3,6): Exit.

\* \*\*Row 4:\*\* \* (4,1): Bottom part of vertical truck. \* (4,2): Empty. \* (4,3): Empty. \* (4,4): Bottom part of vertical bus. \* (4,5): Empty. \* (4,6): Empty.

\* \*\*Row 5:\*\* \* (5,1): Top part of vertical car (bottom left).

\* (5,2): Empty.

\* (5,3): Empty.

\* (5,4): Empty.

\* (5,5)-(5,6): Horizontal grey car. Wait, let's look at the car in the bottom right area. It's above the bottom bus. It looks like it's in Row 5, Cols 5-6? No, looking at the grid lines, the vertical bus is in Col 4. To its right is Col 5. The car is in Row 5, Cols 5-6? No, it looks like it's in Row 5, Cols 5-6. Wait, let's look at the car at (5,5). It's a small car.

\* Actually, let's look at the car in the bottom right quadrant. It's a horizontal car. It is in Row 5. It spans Cols 5 and 6? No, let's look at the vertical bus in the middle (Col 4). To its right is Col 5. The car is in Row 5. It seems to span Cols 5 and 6. Wait, is there a car at (5,5)-(5,6)?

\* Let's look at the car at the very top right. Vertical car at Col 6, Rows 1-2.

\* Let's look at the car at the bottom left. Vertical car at Col 1, Rows 5-6.

\* Let's look at the bus at the bottom. Horizontal. Row 6. Spans Cols 3, 4, 5. Wait, let's count. 1, 2, 3 squares. Yes, length 3. So (6,3), (6,4), (6,5).

\* Let's look at the car above the bottom bus. It's in Row 5. It looks like it's in Cols 5 and 6? No, let's look at the grid.

\* The vertical bus is in Col 4.

\* The car to its right is in Row 5. It looks like it occupies (5,5) and (5,6)? No, looking at the right edge, the exit is at Row 3. The car below the exit area...

\* Let's re-examine the whole grid structure.

\* Columns: 1, 2, 3, 4, 5, 6.

\* Rows: 1, 2, 3, 4, 5, 6.

\* \*\*Left Column (Col 1):\*\*

\* (1,1): Empty. \* (2,1), (3,1), (4,1): Vertical Truck (Length 3). \* (5,1), (6,1): Vertical Car (Length 2).

\* \*\*Col 2:\*\*

\* (1,2), (1,3): Horizontal Car (Top). Wait, (1,2) and (1,3). \* (2,2): Empty. \* (3,2), (3,3): Black Car. Wait, if the top car is (1,2)-(1,3), then the black car is below it. \* Wait, the black car is at (3,2)-(3,3). \* So (2,2) is empty.

\* \*\*Col 3:\*\*

\* (1,3) is occupied by the top car. \* (3,3) is occupied by the black car. \* (6,3), (6,4), (6,5): Bottom Bus. Wait, let's look at the bottom bus. It starts at Col 3? Yes.

\* \*\*Col 4:\*\*

\* (2,4), (3,4), (4,4): Vertical Bus (Length 3). \* (6,4) is occupied by the bottom bus.

\* \*\*Col 5:\*\*

\* (5,5), (5,6): Horizontal Car? Let's look at the car in the lower right. \* It is in Row 5. It looks like it spans (5,5) and (5,6)? No, let's look at the grid lines. \* The vertical bus is in Col 4. \* To its right is Col 5. \* The car is in Row 5. It seems to be in (5,5) and (5,6)? No, looking at the right edge, there is a column 6. \* Wait, let's look at the top right car. It's in Col 6. \* So there is a Col 6. \* The car in Row 5 seems to be in Cols 5 and 6? Or 4 and 5? \* Let's look at

the car above the bottom bus. The bottom bus is in Row 6, Cols 3-5 (length 3). \* So (6,3), (6,4), (6,5) are occupied. \* The car above it is in Row 5. It looks like it's in Cols 5 and 6? No, let's look at the alignment. \* The vertical bus is in Col 4. \* The car to its right is in Row 5. It looks like it starts at Col 5. So (5,5) and (5,6). Wait, is it length 2? Yes, looks like a standard car. \* But wait, look at the top right car. It's vertical in Col 6, Rows 1-2. \* So (5,6) and (6,6) would be empty? \* Let's re-examine the car in Row 5. It looks like it's in Cols 5 and 6? No, looking closely at the image, the car in the lower right (horizontal) is in Row 5. It spans Cols 5 and 6? No, let's look at the grid lines again. \* Grid lines are faint. \* Let's assume standard Rush Hour layout. \* Let's count squares for the bottom bus. It has 5 windows. That usually means length 3 (bus/truck). It spans 3 grid units. It looks like it's in Cols 3, 4, 5. \* So (6,3), (6,4), (6,5) are occupied. \* The car above it (Row 5) looks like it's in Cols 5 and 6? No, that would overlap with the bus if the bus is at 3-5. No, row 5 is above row 6. So no overlap. \* Is the car in Row 5 at Cols 5-6? Or 4-5? \* Let's look at the vertical bus in the middle (Col 4). It ends at Row 4. So (5,4) is empty. \* The car in Row 5 starts to the right of the empty space (5,4). So it starts at (5,5). It looks like it ends at (5,6). So it's a car at (5,5)-(5,6). \* Wait, let's look at the top right car again. Vertical. Col 6. Rows 1-2. \* So (3,6) is the exit. (4,6) is empty. (5,6) is occupied by the horizontal car? If so, the horizontal car is at (5,5)-(5,6). \* Let's check the bottom left car. Vertical. Col 1. Rows 5-6. \* So (5,1) and (6,1) are occupied.

**\*\*3. Determine the path for the black car:\*\***

\* The black car is at (3,2)-(3,3). \* It needs to move Right to exit at (3,6). \* Path: (3,4), (3,5), (3,6). \* Obstacles in the path: \* (3,4) is occupied by the **\*\*Vertical Bus\*\*** (middle one). This bus occupies (2,4), (3,4), (4,4). \* (3,5) is empty? Let's check. \* The vertical bus is in Col 4. \* The exit is at the end of Row 3. \* So the black car needs to clear (3,4) and (3,5). \* Wait, is there a vehicle at (3,5)? \* Looking at the grid, to the right of the middle vertical bus (Col 4) is Col 5. \* In Row 3, Col 5 looks empty. \* In Row 3, Col 6 is the exit. \* So the main blocker is the **\*\*Vertical Bus\*\*** at Col 4, Rows 2-4.

**\*\*4. Determine how to move the blockers:\*\***

\* **\*\*Primary Blocker:\*\*** The Vertical Bus at (2,4)-(4,4).

\* To let the black car pass, this bus must move. \* It can move Up or Down. \* **\*\*Up:\*\*** Blocked by... what's above it? \* (1,4) is empty. \* So it can move Up into (1,4). \* If it moves up 1 square, it occupies (1,4), (2,4), (3,4). Still blocks Row 3. \* It needs to move up 2 squares to clear Row 3. \* Can it move up 2 squares? \* Target positions: (1,4), (2,4), (3,4)... wait. \* Current: (2,4), (3,4), (4,4). \* Move Up 1: (1,4), (2,4), (3,4). Still blocks (3,4). \* Move Up 2: Not possible, grid ends at Row 1. Wait. \* If it moves up, the bottom clears (4,4). The top is at (1,4). \* Current top is (2,4). Move up 1 -> top at (1,4). Bottom at (3,4). Still blocks row 3. \* So it cannot move Up enough to clear Row 3. \* Wait, let's re-evaluate the bus position. \* Maybe it's shorter? No, looks like a bus (3 units). \* Maybe the grid is bigger? No, standard is 6x6. \* Let's look at the top of the bus. It's in Row 2. \* Let's look at the bottom of the bus. It's in Row 4. \* So it occupies Rows 2, 3, 4. \* To clear Row 3, it must move. \* If it moves Up: Top goes to Row 1. Bottom goes to Row 3. Still blocks Row 3. \* So it **\*\*must move Down\*\***.

\* **\*\*Down:\*\*** \* Current bottom is (4,4). \* Below it is (5,4) and (6,4). \* (5,4) looks empty. \* (6,4) is occupied by the **\*\*Bottom Horizontal Bus\*\***. \* The Bottom Horizontal Bus is at Row 6, Cols 3-5 (approx). \* So (6,4) is occupied. \* Therefore, the Middle Vertical Bus cannot move Down because the Bottom Horizontal Bus is in the way.

\* **\*\*Secondary Blocker:\*\*** The Bottom Horizontal Bus (Row 6, Cols 3-5).

\* To let the Middle Vertical Bus move Down, this bus must move. \* It can move Left or Right. \* **\*\*Right:\*\*** Blocked by... \* It's at Cols 3-5. \* To its right is Col 6. \* Is (6,6) empty? \* Let's check the bottom right corner. \* Above it is the horizontal car at (5,5)-(5,6)? Or (5,5) only? \* Let's assume the horizontal car in Row 5 is at (5,5)-(5,6). \* Then (6,6) is empty. \* So the Bottom Horizontal Bus can move Right by 1 square. \* If it moves Right 1 square: Occupies (6,4), (6,5), (6,6). \* Wait, it's currently at (6,3)-(6,5). \* If it moves Right, it goes to (6,4)-(6,6). \* This doesn't clear (6,4). It just shifts the block. \* Wait, if it's at (6,3)-(6,5), then (6,4) is occupied. \* To clear (6,4), it must move Left.

\* **\*\*Left:\*\*** \* Current left is (6,3). \* To its left is (6,2) and (6,1). \* (6,1) is occupied by the **\*\*Bottom Left Vertical Car\*\***. \* (6,2) is empty. \* So it can move Left by 1 square. \* If it moves Left 1 square: Occupies (6,2), (6,3), (6,4). \* Still occupies (6,4). \* Wait, let's look at the length of the bottom bus again. \* It has 5 windows. Usually length 3. \* Let's count grid squares. \* Left wheel is at col 3 start. Right wheel is at col 5 end. So it spans 3, 4, 5. \* So it occupies (6,3), (6,4), (6,5). \* To clear (6,4), it needs to move. \* If it moves Left: Needs to clear (6,4). So it must move to (6,1)-(6,3)? \* (6,1) is occupied by the vertical car. \* So it can move Left to (6,2)-(6,4). Still blocks (6,4). \* Wait, the Middle Vertical Bus needs to move into (5,4) and (6,4)? \* Current Middle Vertical Bus: (2,4), (3,4), (4,4).

\* To clear Row 3, it needs to move Down. \* It needs to move at least 1 square down? \* Move Down 1: (3,4), (4,4), (5,4). Still blocks (3,4). \* Move Down 2: (4,4), (5,4), (6,4). Clears (3,4)!

\* So the Middle Vertical Bus needs to move Down 2 squares. \* This requires (5,4) and (6,4) to be empty. \* Currently: \* (5,4) is empty. \* (6,4) is occupied by the Bottom Horizontal Bus.

\* So the Bottom Horizontal Bus must move out of (6,4). \* The Bottom Horizontal Bus is at (6,3)-(6,5). \* It needs to move Left or Right to clear (6,4). \* If it moves Left: To (6,1)-(6,3)? \* (6,1) is occupied by the vertical car (Rows 5-6). \* So it can't move there. \* Can it move to (6,2)-(6,4)? No, that still has (6,4). \* Wait, if the bus is length 3, and occupies 3,4,5. \* To clear 4, it must move to 1,2,3 or 2,3,4 (no) or ... wait. \* If it moves Left to (6,1)-(6,3), it clears (6,4) and (6,5). \* But (6,1) is occupied. \* So it cannot move Left enough to clear (6,4).

\* If it moves Right: To (6,4)-(6,6)? \* Still occupies (6,4). \* Wait, if it moves Right, it occupies 4, 5, 6. Still blocks 4. \* So the Bottom Horizontal Bus is a problem. \* Maybe my position analysis is wrong.

Let's restart the position analysis carefully. Grid 6x6. \*\*Vehicles:\*\*

1. \*\*Top Horizontal Car:\*\* Row 1, Cols 2-3. (Grey) 2. \*\*Left Vertical Truck:\*\* Col 1, Rows 2-4. (Grey, Length 3) 3. \*\*Bottom Left Vertical Car:\*\* Col 1, Rows 5-6. (Grey, Length 2) 4. \*\*Black Car:\*\* Row 3, Cols 2-3. (Black, Length 2) -> Wait, looking at the image, the black car is to the right of the left truck. \* Left truck is Col 1. \* Black car is in Cols 2-3. \* Wait, is the black car length 2? Yes. \* So it occupies (3,2) and (3,3).

5. \*\*Middle Vertical Bus:\*\* Col 4, Rows 2-4. (Grey, Length 3) 6. \*\*Top Right Vertical Car:\*\* Col 6, Rows 1-2. (Grey, Length 2) 7. \*\*Bottom Horizontal Bus:\*\* Row 6, Cols 3-5. (Grey, Length 3). Wait, let's look at the wheels. \* Left wheel at Col 3. Right wheel at Col 5. \* So it spans Cols 3, 4, 5. \* Wait, is it Cols 2-5? No, looks like 3 units. \* Let's assume Cols 3, 4, 5.

8. \*\*Lower Right Horizontal Car:\*\* Row 5, Cols 5-6? \* Let's look at the car above the bottom bus. \* It's in Row 5. \* It's to the right of the middle vertical bus (Col 4). \* So it starts at Col 5. \* It looks like a standard car (Length 2). \* So it occupies (5,5) and (5,6). \* Wait, if it's at (5,6), and the Top Right Vertical Car is at (1,6)-(2,6). \* Then (3,6) is exit. (4,6) is empty. (5,6) is occupied. (6,6) is empty.

\*\*Goal:\*\* Move Black Car (3,2)-(3,3) to Exit (3,6). \*\*Path:\*\* Needs (3,4), (3,5), (3,6) to be clear. \*\*Current Obstacles in Row 3:\*\*

\* (3,4) is occupied by Middle Vertical Bus. \* (3,5) is empty. \* (3,6) is exit.

\*\*Step 1:\*\* Clear (3,4).

\* Move Middle Vertical Bus (Col 4, Rows 2-4). \* Must move Down to clear Row 3. \* Needs to move Down at least 1 square? \* Current: Rows 2, 3, 4. \* Move Down 1: Rows 3, 4, 5. Still blocks Row 3. \* Move Down 2: Rows 4, 5, 6. Clears Row 3!

\* So Middle Vertical Bus must move to Rows 4, 5, 6 in Col 4. \* This requires (5,4) and (6,4) to be empty.

\*\*Check (5,4) and (6,4):\*\*

\* (5,4): Currently empty. (Middle bus moves into it). \* (6,4): Currently occupied by Bottom Horizontal Bus (Row 6, Cols 3-5). \* So Bottom Horizontal Bus must move.

\*\*Step 2:\*\* Clear (6,4) for the Middle Vertical Bus.

\* Move Bottom Horizontal Bus (Row 6, Cols 3-5). \* Can it move Left? \* Left of Col 3 is Col 2. \* (6,2) is empty. \* (6,1) is occupied by Bottom Left Vertical Car (Rows 5-6). \* So Bus can move Left to Cols 2-4? \* If it moves Left 1 square: Occupies (6,2), (6,3), (6,4). Still blocks (6,4). \* If it moves Left 2 squares: Occupies (6,1), (6,2), (6,3). \* But (6,1) is occupied.

\* So it cannot move Left enough to clear (6,4).

\* Can it move Right? \* Right of Col 5 is Col 6. \* (6,6) is empty. \* So Bus can move Right 1 square. \* New position: Cols 4, 5, 6. \* This still occupies (6,4). \* Wait, if it moves Right, it occupies 4, 5, 6. \* The Middle Vertical Bus wants to move into (6,4). \* So moving Right doesn't help. It keeps (6,4) occupied. \* Wait, if the bus is length 3. \* Current: 3, 4, 5. \* Move Right 1: 4, 5, 6. (6,4) is still occupied. \* So the Bottom Horizontal Bus is completely blocking column 4 at the bottom? \* Let's re-examine the Bottom Horizontal Bus position. \* Maybe it's shorter? Length 2? \*

It has 5 windows. That's long. Definitely length 3. \* Maybe it's further left? \* Let's look at the wheels. \* Left wheel is under the grid line between Col 2 and 3? No, looks like it's in Col 3. \* Right wheel is in Col 5. \* So it spans 3, 4, 5. \* This seems correct.

**\*\*Is there another way?\*\***

\* Maybe the Middle Vertical Bus doesn't need to move all the way down? \* To clear (3,4), the bus (Rows 2-4) must move. \* If it moves Up: \* Top is at Row 2. Row 1 is above it. \* (1,4) is empty. \* Move Up 1: Rows 1, 2, 3. Still blocks (3,4). \* So it MUST move Down.

\* To move Down, it needs space in Rows 5 and 6 (since it's length 3). \* Wait, if it moves Down 1 square: Rows 3, 4, 5. Blocks (3,4). \* If it moves Down 2 squares: Rows 4, 5, 6. Clears (3,4). \* So it MUST occupy (6,4).

\* So (6,4) MUST be cleared. \* The Bottom Horizontal Bus is at (6,3)-(6,5). \* It MUST move. \* If it moves Left: \* Needs to clear (6,4). \* So it must move to (6,1)-(6,3). \* But (6,1) is occupied by the Bottom Left Vertical Car. \* So the Bottom Left Vertical Car must move.

\* **\*\*Step 3:\*\*** Move Bottom Left Vertical Car (Col 1, Rows 5-6). \* Can it move Up? \* Above it is the Left Vertical Truck (Rows 2-4). \* (4,1) is occupied by the truck. \* So it cannot move Up.

\* Can it move Down? \* It's at the bottom edge (Row 6). Cannot move Down.

\* So the Bottom Left Vertical Car is stuck? \* Wait, let's look at the Left Vertical Truck again. \* It's at Col 1, Rows 2-4. \* Can it move? \* Up: (1,1) is empty. So it can move Up 1 square. \* If Left Vertical Truck moves Up 1 square: \* New position: Rows 1-3. \* This clears (4,1).

\* Then Bottom Left Vertical Car (Rows 5-6) can move Up? \* (4,1) is now empty. \* So Bottom Left Vertical Car can move Up 1 square to Rows 4-5. \* This clears (6,1).

\* Now (6,1) is empty. \* Now Bottom Horizontal Bus (Cols 3-5) can move Left? \* It needs to clear (6,4). \* So it needs to move to Cols 1-3? Or 2-4? \* If it moves to Cols 2-4: Still blocks (6,4). \* If it moves to Cols 1-3: Clears (6,4). \* Can it move to Cols 1-3? \* Left is Col 1. (6,1) is now empty (because Bottom Left Vertical Car moved up). \* So yes, it can move Left 2 squares. \* Wait, can it move 2 squares? \* Current: 3, 4, 5. \* Move Left 1: 2, 3, 4. (Blocks 4). \* Move Left 2: 1, 2, 3. (Clears 4). \* Yes, if (6,1) and (6,2) are empty. \* (6,1) is empty (car moved up). \* (6,2) is empty. \* So Bottom Horizontal Bus can move to (6,1)-(6,3).

\* Now (6,4) is empty. \* Now Middle Vertical Bus (Cols 4, Rows 2-4) can move Down. \* It needs to move Down 2 squares to Rows 4-6. \* (5,4) is empty. \* (6,4) is empty. \* So it can move Down 2 squares.

\* Now (3,4) is empty. \* Now Black Car can exit!

**\*\*Let's count the grey vehicles moved:\*\***

1. **\*\*Left Vertical Truck\*\*** (Col 1, Rows 2-4). Move Up 1. \* Is this necessary? \* To move Bottom Left Vertical Car Up, we need space at (4,1). \* Currently (4,1) is occupied by Left Vertical Truck. \* So yes, Left Vertical Truck must move.

2. **\*\*Bottom Left Vertical Car\*\*** (Col 1, Rows 5-6). Move Up 1 (or 2?). \* To clear (6,1) for the Bottom Horizontal Bus. \* Bottom Horizontal Bus needs to move to (6,1)-(6,3) to clear (6,4). \* So (6,1) must be empty. \* So Bottom Left Vertical Car must move Up at least 1 square. \* Can it move Up? Only if (4,1) is clear. \* So Left Vertical Truck moves Up -> clears (4,1). \* Bottom Left Vertical Car moves Up -> clears (6,1).

3. **\*\*Bottom Horizontal Bus\*\*** (Row 6, Cols 3-5). Move Left 2 squares. \* To clear (6,4). \* Wait, does it need to move 2 squares? \* Current: 3, 4, 5. \* Target: Clear (6,4). \* If it moves Left 1: 2, 3, 4. (6,4) still occupied. \* If it moves Left 2: 1, 2, 3. (6,4) clear. \* So yes, it must move.

4. **\*\*Middle Vertical Bus\*\*** (Col 4, Rows 2-4). Move Down 2 squares. \* To clear (3,4). \* This is the direct blocker. \* So yes, it must move.

**\*\*Are there other vehicles?\*\***

\* Top Horizontal Car (Row 1, Cols 2-3). \* Does it need to move? \* Left Vertical Truck moves Up to Row 1. \* Left Vertical Truck is in Col 1. \* Top Horizontal Car is in Cols 2-3. \* So no collision.

\* Top Right Vertical Car (Col 6, Rows 1-2). \* Does it need to move? \* Black car exits through (3,6). \* Path is clear

once (3,4) is clear. (3,5) is already clear. \* So no need to move this car.

\* Lower Right Horizontal Car (Row 5, Cols 5-6). \* Does it need to move? \* Middle Vertical Bus moves Down to Rows 4-6. \* So it occupies (4,4), (5,4), (6,4). \* Lower Right Horizontal Car is at (5,5)-(5,6). \* No collision. \* Wait, let's check the position of Lower Right Horizontal Car again. \* Is it at (5,5)-(5,6)? \* If Middle Vertical Bus moves to (5,4), it's next to (5,5). No problem. \* So this car doesn't need to move.

\*\*So the sequence of moves involves:\*\*

1. Left Vertical Truck (Grey) -> Move Up. 2. Bottom Left Vertical Car (Grey) -> Move Up. 3. Bottom Horizontal Bus (Grey) -> Move Left. 4. Middle Vertical Bus (Grey) -> Move Down.

That's 4 grey vehicles.

\*\*Can we optimize?\*\*

\* Is there a way to clear (6,4) without moving the Bottom Left Vertical Car? \* Bottom Horizontal Bus is at (6,3)-(6,5). \* To clear (6,4), it must move Left to (6,1)-(6,3) or Right to (6,4)-(6,6) [Impossible, still blocks] or Right to (6,5)-(6,7) [Impossible]. \* Wait, if it moves Right... \* Current: 3, 4, 5. \* Move Right 1: 4, 5, 6. (6,4) occupied. \* Move Right 2: 5, 6, 7. (Off grid). \* So it MUST move Left. \* To clear (6,4), it must move Left at least 2 squares (to 1,2,3). \* This requires (6,1) and (6,2) to be empty. \* (6,2) is empty. \* (6,1) is occupied by Bottom Left Vertical Car. \* So Bottom Left Vertical Car MUST move.

\* To move Bottom Left Vertical Car (Rows 5-6): \* Must move Up (can't move Down). \* Requires (4,1) to be empty. \* (4,1) is occupied by Left Vertical Truck. \* So Left Vertical Truck MUST move.

\* To move Left Vertical Truck (Rows 2-4): \* Must move Up (can't move Down, blocked by car below? No, car below is at 5-6. So space at 4 is occupied by truck itself. Space at 5 is car. So truck can't move down). \* Wait, Truck is Rows 2-4. Car is Rows 5-6. \* So there is a gap? No, (4,1) is truck bottom. (5,1) is car top. They are adjacent. \* So Truck cannot move Down. \* Must move Up. \* (1,1) is empty. \* So Truck moves Up 1 square to Rows 1-3. \* This clears (4,1).

\* So the chain is: 1. Move Left Vertical Truck Up. (Grey vehicle 1) 2. Move Bottom Left Vertical Car Up. (Grey vehicle 2) 3. Move Bottom Horizontal Bus Left. (Grey vehicle 3) 4. Move Middle Vertical Bus Down. (Grey vehicle 4) 5. Move Black Car Right. (Target)

\*\*Wait, let me double check the "Minimum number" part.\*\*

\* Do all these vehicles \_need\_ to move? \* "Minimum number of grey vehicles that need to move at least a little bit". \* My sequence involves 4 vehicles. \* Is there a shorter path? \* Maybe the Middle Vertical Bus doesn't need to move all the way down? \* It blocks (3,4). \* It occupies (2,4), (3,4), (4,4). \* To clear (3,4), it must move Up or Down. \* Up: Needs (1,4). \* Move Up 1: (1,4), (2,4), (3,4). Still blocks (3,4). \* So Up is impossible.

\* Down: Needs (5,4), (6,4). \* Move Down 1: (3,4), (4,4), (5,4). Still blocks (3,4). \* Move Down 2: (4,4), (5,4), (6,4). Clears (3,4). \* So it MUST move Down 2 squares. \* This requires (5,4) and (6,4) to be empty.

\* (5,4) is empty. \* (6,4) is occupied by Bottom Horizontal Bus. \* So Bottom Horizontal Bus MUST move. \* Bottom Horizontal Bus is at (6,3)-(6,5). \* To clear (6,4), it must move Left to (6,1)-(6,3) or Right... wait. \* If it moves Right to (6,4)-(6,6), (6,4) is still occupied. \* So it MUST move Left. \* To clear (6,4), it must move Left at least 2 units (to 1-3). \* This requires (6,1) to be empty.

\* (6,1) is occupied by Bottom Left Vertical Car. \* So Bottom Left Vertical Car MUST move. \* Bottom Left Vertical Car is at (5,1)-(6,1). \* To move, it must go Up (blocked by Left Vertical Truck at (4,1)) or Down (blocked by edge). \* So Left Vertical Truck MUST move Up to clear (4,1). \* Left Vertical Truck is at (2,1)-(4,1). \* Move Up 1 to (1,1)-(3,1). \* (1,1) is empty. \* So this is possible.

So it seems 4 vehicles must move.

1. Left Vertical Truck (Col 1, Rows 2-4) -> Moves Up. 2. Bottom Left Vertical Car (Col 1, Rows 5-6) -> Moves Up. 3. Bottom Horizontal Bus (Row 6, Cols 3-5) -> Moves Left. 4. Middle Vertical Bus (Col 4, Rows 2-4) -> Moves Down.

Let's check the other vehicles again.

\* Top Horizontal Car (Row 1, Cols 2-3). \* Left Vertical Truck moves to (1,1)-(3,1). \* Top Horizontal Car is at (1,2)-(1,3). \* (1,1) is next to (1,2). No collision.

\* Top Right Vertical Car (Col 6, Rows 1-2). \* Not in the way.

\* Lower Right Horizontal Car (Row 5, Cols 5-6). \* Middle Vertical Bus moves to (4,4)-(6,4). \* Lower Right Car is at (5,5)-(5,6). \* (5,4) is next to (5,5). No collision.

So the count is 4.

\*\*Wait, let me look really closely at the Bottom Horizontal Bus.\*\*

\* Is it possible that it's shorter? \* It has windows: 1, 2, 3, 4, 5. That's 5 windows. \* The Middle Vertical Bus has windows: 1, 2, 3, 4. That's 4 windows. Wait. \* Let's count windows on Middle Vertical Bus. \* Top to bottom: 1, 2, 3, 4 squares? \* Let's count grid squares. \* Top is at Row 2 line. Bottom is at Row 5 line? \* Let's look at the grid lines again. \* Row 1: Top car. \* Row 2: Top of Left Truck, Top of Middle Bus. \* Row 3: Black Car, Middle of Left Truck, Middle of Middle Bus. \* Row 4: Bottom of Left Truck, Bottom of Middle Bus. \* So Left Truck and Middle Bus are Length 3. (Rows 2, 3, 4). \* This matches my previous analysis.

\* Bottom Horizontal Bus: \* It's in Row 6. \* Left wheel at Col 3. Right wheel at Col 5. \* Wait, let's count windows. 1, 2, 3, 4, 5. \* If it has 5 windows, maybe it's Length 5? No, grid is 6x6. \* Maybe it's Length 4? \* Let's look at the wheels. \* Left wheel is in Col 3. \* Right wheel is in Col 5. \* So it spans Cols 3, 4, 5. Length 3. \* Why 5 windows? Maybe small windows. \* Let's assume Length 3. \* If it's Length 4 (Cols 2-5)? \* Left wheel would be in Col 2. \* Looking at the image, the left wheel is aligned with the vertical line between Col 2 and 3? No, it's inside Col 3. \* So it starts at Col 3. \* Right wheel is inside Col 5. \* So it ends at Col 5. \* So Length 3. (3, 4, 5).

\*\*Let's check the Bottom Left Vertical Car.\*\*

\* Top wheel in Row 5. Bottom wheel in Row 6. \* So Length 2. (Rows 5, 6). \* This matches.

\*\*Let's check the Left Vertical Truck.\*\*

\* Top wheel in Row 2. Bottom wheel in Row 4. \* So Length 3. (Rows 2, 3, 4). \* This matches.

\*\*Let's check the Middle Vertical Bus.\*\*

\* Top wheel in Row 2. Bottom wheel in Row 4. \* So Length 3. (Rows 2, 3, 4). \* This matches.

\*\*Let's check the Lower Right Horizontal Car.\*\*

\* It's in Row 5. \* Left wheel in Col 5. Right wheel in Col 6. \* So Length 2. (Cols 5, 6). \* Wait, looking at the image... \* The car is to the right of the Middle Vertical Bus (Col 4). \* So it starts at Col 5. \* It ends at the right edge? \* The right edge is Col 6. \* So it's at (5,5)-(5,6). \* This matches.

\*\*Let's check the Top Right Vertical Car.\*\*

\* Top wheel in Row 1. Bottom wheel in Row 2. \* So Length 2. (Rows 1, 2). \* This matches.

\*\*Let's check the Top Horizontal Car.\*\*

\* Left wheel in Col 2. Right wheel in Col 3. \* So Length 2. (Cols 2, 3). \* This matches.

\*\*So the analysis holds.\*\* Number of grey vehicles to move: 4.

\*\*Is there any alternative?\*\*

\* What if the Middle Vertical Bus moves Up?

\* It's at Rows 2-4. \* To clear Row 3, it must move Up so that its bottom is above Row 3. \* Bottom is at Row 4. \* Move Up 1: Bottom at Row 3. Still blocks. \* Move Up 2: Bottom at Row 2. Top at Row 0 (off grid). Impossible. \* So Middle Vertical Bus MUST move Down.

\* What if the Bottom Horizontal Bus moves Right?

\* It's at Cols 3-5. \* Move Right 1: Cols 4-6. \* (6,4) is still occupied. \* So Middle Vertical Bus cannot move into (6,4).

\* So Bottom Horizontal Bus MUST move Left.

\* What if Bottom Horizontal Bus moves Left 1 square?

\* New pos: Cols 2-4. \* (6,4) is still occupied. \* So it must move Left 2 squares. \* New pos: Cols 1-3. \* This requires (6,1) to be empty.

\* What if Bottom Left Vertical Car moves Up 1 square?

\* New pos: Rows 4-5. \* This requires (4,1) to be empty. \* (4,1) is occupied by Left Vertical Truck. \* So Left Vertical Truck must move.

\* What if Left Vertical Truck moves Up 1 square?

\* New pos: Rows 1-3. \* (1,1) is empty. \* This works.

So the chain is unbreakable.

1. Left Vertical Truck (Grey) 2. Bottom Left Vertical Car (Grey) 3. Bottom Horizontal Bus (Grey) 4. Middle Vertical Bus (Grey)

Total: 4.

Wait, let's look at the "Lower Right Horizontal Car" again.

\* Is it possible that the Middle Vertical Bus moves down, and pushes into (5,4), and the Lower Right Car is at (5,5)-(5,6). \* Yes, that's fine. \* But the Middle Vertical Bus needs to go to Row 6 to clear Row 3. \* So it occupies (4,4), (5,4), (6,4). \* This is fine with Lower Right Car at (5,5).

\*\*Is it possible that I misidentified a vehicle?\*\*

\* "All vehicles in the garage can only drive forwards or backwards." \* "The black car wants to leave the garage." \* "What is the minimum number of grey vehicles that need to move at least a little bit so that this is possible?"

Let's double check the grid again. Maybe I missed a gap.

\* Look at (5,4). Is it empty? \* Middle Vertical Bus is at (2,4)-(4,4). \* Bottom Horizontal Bus is at (6,3)-(6,5). \* Lower Right Car is at (5,5)-(5,6). \* So (5,4) is indeed empty.

\* Look at (3,5). Is it empty? \* Middle Vertical Bus is at Col 4. \* Lower Right Car is at Row 5. \* Top Right Car is at Col 6. \* So (3,5) is empty. \* Wait, is there a car at (3,5)? \* Looking at the grid, to the right of the Middle Vertical Bus (Col 4) is Col 5. \* In Row 3, Col 5 looks empty. \* So the path for the black car is (3,4) -> (3,5) -> (3,6). \* Only (3,4) is blocked.

So the logic holds. Blocker: Middle Vertical Bus. To move it Down: Need (6,4) clear. Blocker for (6,4): Bottom Horizontal Bus. To move it Left: Need (6,1) clear. Blocker for (6,1): Bottom Left Vertical Car. To move it Up: Need (4,1) clear. Blocker for (4,1): Left Vertical Truck. To move it Up: Need (1,1) clear. (It is clear).

So the chain is:

1. Move Left Vertical Truck Up. (Grey vehicle 1) 2. Move Bottom Left Vertical Car Up. (Grey vehicle 2) 3. Move Bottom Horizontal Bus Left. (Grey vehicle 3) 4. Move Middle Vertical Bus Down. (Grey vehicle 4) 5. Move Black Car Right.

Are there any other dependencies?

\* When Left Vertical Truck moves Up to (1,1)-(3,1)... \* Does it block anything? \* Top Horizontal Car is at (1,2)-(1,3). \* (1,1) is adjacent to (1,2). No block.

\* When Bottom Left Vertical Car moves Up to (4,1)-(5,1)... \* (4,1) was occupied by Left Vertical Truck. Now empty. \* (5,1) was occupied by Bottom Left Vertical Car. Now empty? No, it moves into (4,1)-(5,1). \* Wait, if it moves Up 1 square: \* Old: (5,1)-(6,1). \* New: (4,1)-(5,1).

\* This clears (6,1). \* This works.

So 4 vehicles.

Let me just pause and think if I can move the Bottom Horizontal Bus Right.

\* If Bottom Horizontal Bus moves Right... \* It's at (6,3)-(6,5). \* Move Right 1: (6,4)-(6,6). \* (6,4) is still occupied. \* So Middle Vertical Bus cannot move down into (6,4). \* So this path is dead.

What if Middle Vertical Bus moves Up?

\* It's at (2,4)-(4,4). \* Move Up 1: (1,4)-(3,4). \* (3,4) is still occupied. \* So this path is dead.

So Middle Vertical Bus MUST move Down 2 squares. This requires (5,4) and (6,4) empty. (5,4) is empty. (6,4) is occupied by Bottom Horizontal Bus. So Bottom Horizontal Bus MUST move. To clear (6,4), it must move Left (since moving Right keeps (6,4) occupied). To move Left enough (2 squares to clear col 4), it needs (6,1) and (6,2) empty. (6,2) is empty. (6,1) is occupied by Bottom Left Vertical Car. So Bottom Left Vertical Car MUST move. To move, it must go Up (blocked by edge at bottom). To go Up, it needs (4,1) empty. (4,1) is occupied by Left Vertical Truck. So Left Vertical Truck MUST move. To move, it must go Up (blocked by car below at 5-6... wait).

\* Left Vertical Truck is at (2,1)-(4,1). \* Bottom Left Vertical Car is at (5,1)-(6,1). \* So Truck cannot move Down. \* Truck must move Up. \* (1,1) is empty. \* So Truck moves Up 1 square to (1,1)-(3,1). \* This clears (4,1).

So yes, 4 vehicles.

Let's double check the question. "Minimum number of grey vehicles". The black car is not grey. The vehicles are:

1. Top Horizontal (Grey) 2. Left Vertical Truck (Grey) 3. Bottom Left Vertical Car (Grey) 4. Middle Vertical Bus (Grey) 5. Top Right Vertical Car (Grey) 6. Bottom Horizontal Bus (Grey) 7. Lower Right Horizontal Car (Grey)

My solution moves:

1. Left Vertical Truck 2. Bottom Left Vertical Car 3. Bottom Horizontal Bus 4. Middle Vertical Bus

Are any of these not grey?

\* Left Vertical Truck: Grey. \* Bottom Left Vertical Car: Grey. \* Bottom Horizontal Bus: Grey. \* Middle Vertical Bus: Grey. All are grey.

Is it possible that the Top Horizontal Car needs to move?

\* Left Vertical Truck moves to (1,1)-(3,1). \* Top Horizontal Car is at (1,2)-(1,3). \* They are adjacent. No collision. \* So Top Horizontal Car does not need to move.

Is it possible that the Lower Right Horizontal Car needs to move?

\* Middle Vertical Bus moves to (4,4)-(6,4). \* Lower Right Horizontal Car is at (5,5)-(5,6). \* They are adjacent. No collision. \* So Lower Right Horizontal Car does not need to move.

Is it possible that the Top Right Vertical Car needs to move?

\* Black car exits at (3,6). \* Top Right Vertical Car is at (1,6)-(2,6). \* No collision. \* So Top Right Vertical Car does not need to move.

So the answer is 4.

Let me just re-read the grid to ensure I haven't missed a "sneaky" move. What if the Bottom Horizontal Bus is actually shorter?

\* If it's length 2, at (6,4)-(6,5). \* Then (6,3) is empty. \* Then to clear (6,4), it must move Right to (6,5)-(6,6). \* Then (6,4) is clear. \* Then Middle Vertical Bus can move Down. \* In this case, only 1 vehicle (Bottom Horizontal Bus) needs to move? \* But looking at the image, it has 5 windows and spans 3 grid columns (3, 4, 5). \* The wheels are at the ends. Left wheel in col 3, right wheel in col 5. \* So it's definitely length 3.

What if the Bottom Left Vertical Car is shorter?

\* Length 1? No, cars are length 2 or 3. \* It has 2 windows? No, looks like a car. Length 2. \* Wheels at Row 5 and Row 6. \* So it's length 2.

What if the Left Vertical Truck is shorter?

\* Length 2? \* Wheels at Row 2 and Row 4? No, that would be length 3. \* Wheels at Row 2 and Row 3? \* Let's count windows. 1, 2, 3, 4. \* Wait, the Left Vertical Truck has 4 windows. \* The Middle Vertical Bus has 4 windows. \* The Bottom Horizontal Bus has 5 windows. \* This suggests lengths. \* Usually in Rush Hour: \* Car = Length 2. \* Truck/Bus = Length 3.

\* Left Vertical Truck: 4 windows? Maybe it's length 3. \* If it's length 3, it occupies 3 cells. \* Windows are just decorative. \* Let's assume standard Rush Hour rules. \* Left Vertical object: Looks like a truck. Length 3. \* Middle Vertical object: Looks like a bus. Length 3. \* Bottom Horizontal object: Looks like a limo/bus. Length 3. \* Others are cars (Length 2).

Let's assume my length analysis is correct. Left Vertical Truck: Length 3 (Rows 2-4). Bottom Left Vertical Car: Length 2 (Rows 5-6). Bottom Horizontal Bus: Length 3 (Cols 3-5). Middle Vertical Bus: Length 3 (Rows 2-4).

Is it possible that the Left Vertical Truck is actually Length 2?

\* If it's Length 2, where is it? \* Top at Row 2. Bottom at Row 3. \* Then (4,1) is empty. \* Then Bottom Left Vertical Car (Rows 5-6) can move Up to (4,1)-(5,1)? \* Wait, if Truck is at (2,1)-(3,1). \* Then (4,1) is empty. \* Bottom Left Car is at (5,1)-(6,1). \* Move Up 1: (4,1)-(5,1). \* Clears (6,1). \* Then Bottom Horizontal Bus can move Left to (6,1)-(6,3). \* Clears (6,4). \* Then Middle Vertical Bus moves Down. \* In this case, vehicles moved: 1. Bottom Left Vertical Car. 2. Bottom Horizontal Bus. 3. Middle Vertical Bus.

\* Left Vertical Truck didn't need to move! \* So the answer would be 3.

Let's look really closely at the Left Vertical Truck.

\* Top wheel is in Row 2. \* Bottom wheel is in Row 4. \* So it spans Rows 2, 3, 4. \* It has 4 rectangular windows. \* If it spans 3 rows, it's Length 3. \* So my initial analysis was correct. It blocks (4,1).

Let's look at the Bottom Left Vertical Car.

\* Top wheel in Row 5. \* Bottom wheel in Row 6. \* So it spans Rows 5, 6. \* It's Length 2. \* So it blocks (6,1).

So the chain stands. Truck (2-4) blocks Car (5-6) from moving up?

\* Truck bottom is at Row 4. \* Car top is at Row 5. \* They are adjacent. \* So Car cannot move Up unless Truck moves Up. \* Truck can move Up to (1,1)-(3,1). \* This clears (4,1). \* Then Car moves Up to (4,1)-(5,1). \* This clears (6,1). \* Then Bus (3-5) moves Left to (1-3). \* This clears (6,4). (Wait, Bus is at Row 6). \* Bus moves Left to (6,1)-(6,3). \* This clears (6,4). \* Then Middle Vertical Bus (2-4) moves Down to (4-6). \* This clears (3,4). \* Black car exits.

So 4 vehicles.

Is there any other path?

\* What about the Top Horizontal Car? \* It's at (1,2)-(1,3). \* Does it block anything? \* No. \* What about Top Right Vertical Car? \* At (1,6)-(2,6). \* Does it block anything? \* No. \* What about Lower Right Horizontal Car? \* At (5,5)-(5,6). \* Does it block anything? \* Middle Vertical Bus moves to (4,4)-(6,4). \* (5,4) is next to (5,5). No block.

So it seems robustly 4.

Let me think if "move at least a little bit" implies something.

\* Yes, even 1 square counts. \* My moves: \* Truck: Up 1 square. \* Car (bottom left): Up 1 square. \* Bus (bottom): Left 2 squares. \* Bus (middle): Down 2 squares.

\* All move "at least a little bit".

Is it possible that the Bottom Horizontal Bus is Length 2?

\* If it's Length 2, at (6,4)-(6,5). \* Then (6,3) is empty. \* Then to clear (6,4), it must move Right to (6,5)-(6,6). \* Then (6,4) is clear. \* Then Middle Vertical Bus moves Down. \* Vehicles moved: 1. Bottom Horizontal Bus. 2. Middle Vertical Bus.

\* Answer: 2. \* But looking at the image, it's clearly longer than the cars. \* The cars (top, black, bottom left, top right, lower right) are all the same size (Length 2). \* The vertical truck (left), vertical bus (middle), horizontal bus (bottom) are longer. \* Left vertical truck: 3 units. \* Middle vertical bus: 3 units. \* Bottom horizontal bus: 3 units. \* So my length analysis is correct.

Is it possible that the Middle Vertical Bus is Length 2?

\* If it's Length 2, at (2,4)-(3,4). \* Then to clear (3,4), it must move Up or Down. \* Move Up: (1,4)-(2,4). Clears (3,4).

\* (1,4) is empty. \* So it can move Up. \* Vehicles moved: 1 (Middle Vertical Bus).

\* Move Down: (3,4)-(4,4). Still blocks. \* Move Down 2: (4,4)-(5,4). Clears (3,4). \* Requires (5,4) empty (yes) and (4,4) empty (yes, if it moves). \* Wait, if it's length 2 at (2,4)-(3,4). \* Move Down 1: (3,4)-(4,4). Blocks. \* Move Down 2: (4,4)-(5,4). Clears. \* This requires (5,4) empty. Yes. \* So if it's Length 2, it can move Up 1 square to clear the path. \* Or move Down 2 squares.

\* But looking at the image, it has 4 windows. \* The cars have 2 windows (side view) or just a shape. \* The black car has a shape. \* The middle bus has 4 distinct rectangular windows. \* The left truck has 4 distinct rectangular windows. \* The bottom bus has 5 distinct rectangular windows. \* This strongly suggests Length 3 (or 4 for the bottom one?). \* If Bottom Bus is Length 4 (Cols 2-5)? \* Left wheel at Col 2? \* Looking at the grid line... \* Col 1 has the vertical car. \* Col 2 is empty in Row 6. \* The bottom bus starts at Col 3. \* So it's Length 3.

So, assuming standard Rush Hour vehicle sizes (2 and 3):

\* Left Vertical: 3. \* Middle Vertical: 3. \* Bottom Horizontal: 3. \* Others: 2.

So the answer 4 seems correct.

Let's double check if I can move the Middle Vertical Bus \_Up\_.

\* It is at Rows 2, 3, 4. \* To clear Row 3, it must move Up so bottom > 3. \* Bottom is 4. \* Move Up 1 -> Bottom 3. Still blocks. \* Move Up 2 -> Bottom 2. Top 0. Impossible. \* So it MUST move Down.

Let's double check if I can move the Bottom Horizontal Bus \_Right\_.

\* It is at Cols 3, 4, 5. \* To clear Col 4, it must move Right so left > 4. \* Left is 3. \* Move Right 1 -> Left 4. Still blocks. \* Move Right 2 -> Left 5. \* New pos: Cols 5, 6, 7. \* Col 7 is off grid. \* So impossible.

\* So it MUST move Left.

Let's double check if I can move the Bottom Left Vertical Car \_Down\_.

\* It is at Rows 5, 6. \* Row 6 is bottom edge. \* Impossible. \* So it MUST move Up.

Let's double check if I can move the Left Vertical Truck \_Down\_.

\* It is at Rows 2, 3, 4. \* Below is Bottom Left Vertical Car at Rows 5, 6. \* They are adjacent (4 and 5). \* So impossible. \* So it MUST move Up.

So the chain is strictly forced.

1. Left Vertical Truck (Up) 2. Bottom Left Vertical Car (Up) 3. Bottom Horizontal Bus (Left) 4. Middle Vertical Bus (Down)

Wait, check the Bottom Horizontal Bus move again.

\* It needs to clear (6,4). \* It is at (6,3)-(6,5). \* Move Left 1 -> (6,2)-(6,4). (6,4) still occupied. \* Move Left 2 -> (6,1)-(6,3). (6,4) clear. \* So it must move Left 2 squares. \* This requires (6,1) and (6,2) to be empty. \* (6,2) is empty. \* (6,1) is occupied by Bottom Left Vertical Car. \* So Bottom Left Vertical Car must move. \* It moves Up 1 square to (4,1)-(5,1). \* This requires (4,1) to be empty. \* (4,1) is occupied by Left Vertical Truck. \* So Left Vertical Truck must move. \* It moves Up 1 square to (1,1)-(3,1). \* This requires (1,1) to be empty. \* (1,1) is empty.

So yes, 4 vehicles.

Is it possible that the "Lower Right Horizontal Car" is actually blocking the Middle Vertical Bus?

\* Middle Vertical Bus moves to (4,4)-(6,4). \* Lower Right Car is at (5,5)-(5,6). \* (5,4) is part of the bus. (5,5) is part of the car. \* They are adjacent. No block.

Is it possible that the "Top Right Vertical Car" is blocking the exit?

\* Exit is Row 3. \* Top Right Car is Rows 1-2. \* No block.

So the answer is definitely 4.

Let's write down the final argument.

1. Identify the goal: Move Black Car (Row 3) to the right exit. 2. Identify immediate obstacle: The vertical grey bus in Column 4 (Rows 2-4) blocks the path at (3,4). 3. Determine how to move the obstacle: The bus must move down to clear Row 3. Moving up is not enough (would still block Row 3). Moving down requires space in Rows 5 and 6 of Column 4. 4. Identify secondary obstacle: The horizontal grey bus in Row 6 (Cols 3-5) occupies (6,4), blocking the vertical bus from moving down. 5. Determine how to move the secondary obstacle: It must move left to clear (6,4). Moving right keeps (6,4) occupied. Moving left requires space in Cols 1 and 2 of Row 6. 6. Identify tertiary obstacle: The vertical grey car in Column 1 (Rows 5-6) occupies (6,1), blocking the horizontal bus from moving left enough. 7. Determine how to move the tertiary obstacle: It must move up to clear (6,1). Moving down is impossible (edge). Moving up requires space in Row 4 of Column 1. 8. Identify quaternary obstacle: The vertical grey truck in Column 1 (Rows 2-4) occupies (4,1), blocking the vertical car from moving up. 9. Determine how to move the quaternary obstacle: It must move up. Space is available at (1,1). 10. Count the vehicles: \* Vertical Truck (Col 1) - Moves Up. \* Vertical Car (Col 1, bottom) - Moves Up. \* Horizontal Bus (Row 6) - Moves Left. \* Vertical Bus (Col 4) - Moves Down. \* Total: 4 grey vehicles.

One final check. Is the "Vertical Truck" actually two vehicles?

\* No, it's a single long vehicle. \* Is the "Bottom Left Vertical Car" actually part of the truck? \* No, there is a grid line between Row 4 and 5. \* The truck ends at Row 4. The car starts at Row 5. \* So they are separate.

Is the "Bottom Horizontal Bus" actually two vehicles?

\* No, looks like one long vehicle.

Is the "Middle Vertical Bus" actually two vehicles?

\* No, looks like one long vehicle.

So the count remains 4.

The question asks for the "minimum number of grey vehicles". My sequence moves 4 grey vehicles. Is it possible to do it with fewer?

\* We established that Middle Vertical Bus MUST move. (1) \* To move it, Bottom Horizontal Bus MUST move. (2) \* To move Bottom Horizontal Bus left enough, Bottom Left Vertical Car MUST move. (3) \* To move Bottom Left Vertical Car up, Left Vertical Truck MUST move. (4) \* Are there any alternatives? \* Could Bottom Horizontal Bus move Right? No, doesn't clear (6,4). \* Could Middle Vertical Bus move Up? No, doesn't clear (3,4). \* Could Bottom Left Vertical Car move Down? No, edge. \* Could Left Vertical Truck move Down? No, blocked by car.

So the chain is necessary. Answer is 4.

Wait, looking at the Top Right Vertical Car. It is in Col 6, Rows 1-2. The exit is Row 3. So it's not blocking.

Looking at the Top Horizontal Car. Row 1, Cols 2-3. Not blocking.

Looking at the Lower Right Horizontal Car. Row 5, Cols 5-6. Not blocking.

So only the 4 vehicles in the chain need to move.

Final Answer seems to be 4.

One tiny detail: "move at least a little bit". My moves:

\* Truck: Up 1 unit. \* Car (bottom left): Up 1 unit. \* Bus (bottom): Left 2 units. \* Bus (middle): Down 2 units. All move > 0.

So 4 vehicles.

Let's double check the grid coordinates one last time. Maybe the Left Vertical Truck is only Rows 2-3?

\* If it's Rows 2-3 (Length 2). \* Then (4,1) is empty. \* Then Bottom Left Vertical Car (Rows 5-6) can move Up to (4,1)-(5,1). \* Then (6,1) is clear. \* Then Bottom Horizontal Bus (3-5) can move Left to (1-3). \* Then (6,4) is clear. \* Then Middle Vertical Bus (2-4) can move Down to (4-6). \* Then (3,4) is clear. \* In this case, Left Vertical Truck does NOT need to move. \* So answer would be 3.

How to determine the length of the Left Vertical Truck?

\* Look at the windows. 1, 2, 3, 4 windows. \* Look at the wheels. Top wheel in Row 2. Bottom wheel in Row 4. \* This implies it spans Rows 2, 3, 4. Length 3. \* Look at the Middle Vertical Bus. 1, 2, 3, 4 windows. Top wheel Row 2. Bottom wheel Row 4. Length 3. \* Look at the Bottom Horizontal Bus. 1, 2, 3, 4, 5 windows. Left wheel Col 3. Right wheel Col 5. Length 3. \* Look at the cars. 2 windows/side. Length 2. \* So the "Trucks/Buses" are Length 3. \* So Left Vertical Truck is Length 3. \* So it occupies (2,1), (3,1), (4,1). \* So it blocks (4,1). \* So my initial analysis holds. Answer is 4.

Just to be absolutely sure, look at the gap between Left Vertical Truck and Bottom Left Vertical Car.

\* Truck bottom is at Row 4 line. \* Car top is at Row 5 line. \* They are touching. \* So Truck is Rows 2-4. Car is Rows 5-6. \* Correct.

Look at the gap between Middle Vertical Bus and Bottom Horizontal Bus.

\* Bus bottom is at Row 4 line. \* Bottom Bus top is at Row 6 line. \* Wait. \* Middle Vertical Bus is Rows 2-4. \* Bottom Horizontal Bus is Row 6. \* So Row 5 is empty in Col 4. \* This matches my analysis. (5,4) is empty. \* So Middle Vertical Bus needs to move into (5,4) and (6,4). \* So it needs to move 2 squares down. \* This requires (6,4) to be empty. \* Correct.

So everything is consistent. Answer: 4.

</think>

The black car is in the third row and needs to exit to the right.

1. **Direct Obstacle:** The vertical grey bus in the fourth column (let's call it the "middle bus") is blocking the black car's path. It occupies rows 2, 3, and 4 of column 4. To let the black car pass, this bus must move. 2. **Moving the Middle Bus:** It cannot move up enough to clear row 3 (moving up 1 square would still leave it in row 3). So, it must move down. To move down 2 squares (clearing row 3), it needs the spaces in row 5 and row 6 of column 4 to be empty. Row 5, column 4 is empty. Row 6, column 4 is occupied by the long horizontal grey bus at the bottom. 3. **Moving the Bottom Bus:** The bottom horizontal bus occupies columns 3, 4, and 5 in row 6. To clear column 4, it must move. Moving right won't help (it would still occupy column 4 or go off-grid). It must move left. To clear column 4, it needs to move left by 2 squares, into columns 1, 2, and 3. 4. **Moving the Bottom-Left Car:** The space in row 6, column 1 is occupied by a vertical grey car. So, the bottom horizontal bus cannot move into that spot yet. This vertical car (occupying rows 5 and 6 of column 1) must move. It can only move up. 5. **Moving the Left Truck:** To move the bottom-left vertical car up, the space in row 4, column 1 must be empty. However, it is occupied by the long vertical grey truck on the left (occupying rows 2, 3, and 4 of column 1). So, this truck must move up. There is space in row 1, column 1, so it can move up 1 square.

So, the chain of moves is:

1. Move the **left vertical truck** up. 2. Move the **bottom-left vertical car** up. 3. Move the **bottom horizontal bus** left. 4. Move the **middle vertical bus** down. 5. Finally, the black car can exit.

There are **4** grey vehicles that need to move.

4

## ## Multilingual Support

With a major leap in multilingual capabilities, Qwen3.5 now supports over 200 languages. This update prioritizes the expansion of low-resource languages. Through this broader linguistic scope, Qwen3.5 is dedicated to fostering global AI equity.

| Language Family | Languages & Dialects | | --- | --- | | Indo-European | English, French, Portuguese, German, Romanian, Swedish, Danish, Bulgarian, Russian, Czech, Greek, Ukrainian, Spanish, Dutch, Slovak, Croatian, Polish, Lithuanian, Norwegian Bokmål, Norwegian Nynorsk, Persian, Slovenian, Gujarati, Latvian, Italian, Occitan, Nepali, Marathi, Belarusian, Serbian, Luxembourgish, Venetian, Assamese, Welsh, Silesian, Asturian, Chhattisgarhi, Awadhi, Maithili, Bhojpuri, Sindhi, Irish, Faroese, Hindi, Punjabi, Bengali, Oriya, Tajik, Eastern Yiddish, Lombard, Ligurian, Sicilian, Friulian, Sardinian, Galician, Catalan, Icelandic, Tosk Albanian, Limburgish, Dari, Afrikaans, Macedonian, Sinhala, Urdu, Magahi, Bosnian, Armenian, **Latgalian**, Scottish Gaelic, Central Kurdish, Northern Kurdish, Southern Pashto, Sanskrit, Dhundari, Marwari, Ahirani, Bagheli, Bagri, Bundeli, Braj, Kumaoni, Kashmiri **\*\*** | | Sino-Tibetan | Chinese (Simplified Chinese, Traditional Chinese, Cantonese), Burmese,

**\*\*Standard Tibetan, Meitei\*\*** | | Afro-Asiatic | Arabic (Standard, Najdi, Levantine, Egyptian, Moroccan, Mesopotamian, Ta'izzi-Adeni, Tunisian, **\*\*Gulf, Algerian, Sudanese, Libyan\*\***), Hebrew, Maltese, **\*\*Amharic, Tigrinya, Kabyle, Somali, West Central Oromo, Hausa\*\*** | | Austronesian | Indonesian, Malay, Tagalog, Cebuano, Javanese, Sundanese, Minangkabau, Balinese, Banjar, Pangasinan, Iloko, Waray (Philippines), **\*\*Plateau Malagasy, Malagasy, Buginese, Maori, Samoan, Hawaiian, Fijian\*\*** | | Dravidian | Tamil, Telugu, Kannada, Malayalam | | Turkic | Turkish, North Azerbaijani, Northern Uzbek, Kazakh, Bashkir, Tatar, **\*\*Crimean Tatar, Kyrgyz, Turkmen, Uyghur\*\*** | | Tai-Kadai | Thai, Lao, **\*\*Shan\*\*** | | Uralic | Finnish, Estonian, Hungarian, **\*\*Meadow Mari\*\*** | | Austroasiatic | Vietnamese, Khmer | | **\*\*Niger–Congo\*\*** | **\*\*Yoruba, Ewe, Kinyarwanda, Lingala, Northern Sotho, Nyanja, Shona, Southern Sotho, Tswana, Xhosa, Zulu, Luganda, Swati, Tsonga, Tumbuka, Venda, Chokwe, Luba-Kasai, Rundi, Umbundu, Kikuyu, Kongo, Nigerian Fulfulde, Wolof, Fon, Kabiye, Mossi, Akan, Twi, Bambara, Igbo\*\*** | | Other | Japanese, Korean, Georgian, Basque, Haitian, Papiamentu, Kabuverdianu, Tok Pisin, Swahili, **\*\*Central Aymara, Tulu, Nagamese, Nigerian Pidgin, Mauritian Creole, Sango, Ayacucho Quechua, Halh Mongolian, Southwestern Dinka, Nuer, Guarani\*\*** |

## ## Citation

Feel free to cite the following article if you find Qwen3.5 helpful:

`@misc{qwen35blog, title = {Qwen3.5: Towards Native Multimodal Agents}, url = {https://qwen.ai/blog?id=qwen3.5}, author = {Qwen Team}, month = {February}, year = {2026}}`