

Description

Operating rooms have lots of knobs and levers to turn. Voice assistants are already common in smart homes, but reliable, real-time control is a different challenge in an operating room. I test JevK5-2B, an open-weight 2B model, running locally on my RTX 2080, and compare it with TypeSafe's Jev on a virtual operating room task. Jev is more accurate and faster in my tests, so this JevK5-2B setup doesn't yet look like a practical local alternative.

Contents

1	System-One and Jev	1
1.1	How do they compare to LLMs?	1
1.2	Local Open-Weight alternatives	1
2	Project introduction	3
2.1	The virtual operating room	3
2.2	From utterance to room state	4
2.3	Defining the choices	7
2.4	Running locally	7
3	Evaluation	9
3.1	Dataset and procedure	9
3.2	Overall results	9
3.3	It depends on the field	10
3.4	Fast enough, on which machine?	11
4	Conclusion	12
4.1	Potential improvements	12
	References	13

1 System-One and Jev

If the term is unfamiliar to you, here is the short version: System 1 is the fast, intuitive, and emotional system, while System 2 is the slow, deliberate, and logical one. Daniel Kahneman, Nobel Prize laureate in Economics, coined this term in his psychology work on judgment and decision making, which challenged the idea that humans are rational decision makers.^{KAHNEMAN, Daniel (2)}

TypeSafe leaned into this and named their new type of classification models “System-One Models”, in contrast to text-generating Large Language Models (LLMs), which are more like System 2. Jev is their first System-One model release.^{ALMEIDA, Diogo (3)}

1.1 How do they compare to LLMs?

LLMs work by predicting the most probable next token, step by step. Especially for machine-to-machine communication, people have tried to squeeze LLM output into a schema. There are techniques for that, but they limit the model in its reasoning capabilities^{ALMEIDA, Diogo (3)}, and they are nowhere near efficient enough for realtime applications at a feasible cost.

So what does Jev do differently? It takes the same input as basic LLMs (which is plain text) and a set of questions with their answer candidates. Its output is then a set of probabilities for each question. The model fundamentally does not generate text and therefore cannot return an answer outside the supplied schema.¹ That reliability and efficiency make it a good fit for realtime applications.^{ALMEIDA, Diogo (3)}

Oh, and they are really really cheap. Jev is currently listed at 0.0047€ per 1M input tokens, with output tokens being free. TypeSafe attributes this price to the massive parallelism of its decision heads.^{ALMEIDA, Diogo (3)}

1.2 Local Open-Weight alternatives

Despite Jev having been released only on September 15th, open-weight alternatives have sprouted up like mushrooms. They mostly work by architecturally modifying and fine-tuning a

Table 1.1: Comparison between System-One and LLMs^{ALMEIDA, Diogo (3)}

	Large Language Model	System-One Model
Fundamental Output	Textstream 	Typed Probabilities 
Reasoning Efforts	High, Long Thinking 	Low, Intuitive 
Cost Efficiency and Speed	Low, Slow 	High, Fast 

¹Which does **not** mean it cannot pick an incorrect answer.

pre-trained language model on a specific dataset. Since the problem space is less fragmented, way smaller models get acceptable results.

Open weights → Great news for privacy

TypeSafe AI claims not to train on your data ^{ALMEIDA, Diogo (3)}, which sounds good on paper. For highly sensitive applications, like handling health data, this is not enough. Open-weight models can be self-hosted, which gives the operator more control over where inference data goes. It does not guarantee that the whole application keeps no data off-premises.

Some notable contenders are:

- [JevK5](#) - 4B params, the open-weight benchmark leader when I started this experiment
- [Kev](#) - Similar design to JevK5
- [DiffusionGemma](#) - ~26B params, based on diffusion
- [Laya](#) - ~500MB size, less than 50ms latency even on CPU, often requires fine-tuning

2 Project introduction

2.1 The virtual operating room

Operating rooms have lots of equipment, and each device brings its own controls. A voice assistant could save you the constant reaching for knobs and menus. But a natural command is not enough on its own. The system has to identify which device and setting the speaker means, handle values precisely, and avoid changing anything the speaker did not ask to change.

So let me quickly cook up a beautiful virtual room for this experiment. This room has an **insufflator**, an operating **light**, a surgical **table**, and an **endoscope**. Each object maps to a handful of state fields, such as gas flow, light brightness, table tilt, or endoscope zoom. Fig. 2.1 shows the room and Tab. 2.1 shows the set of actions for each object.

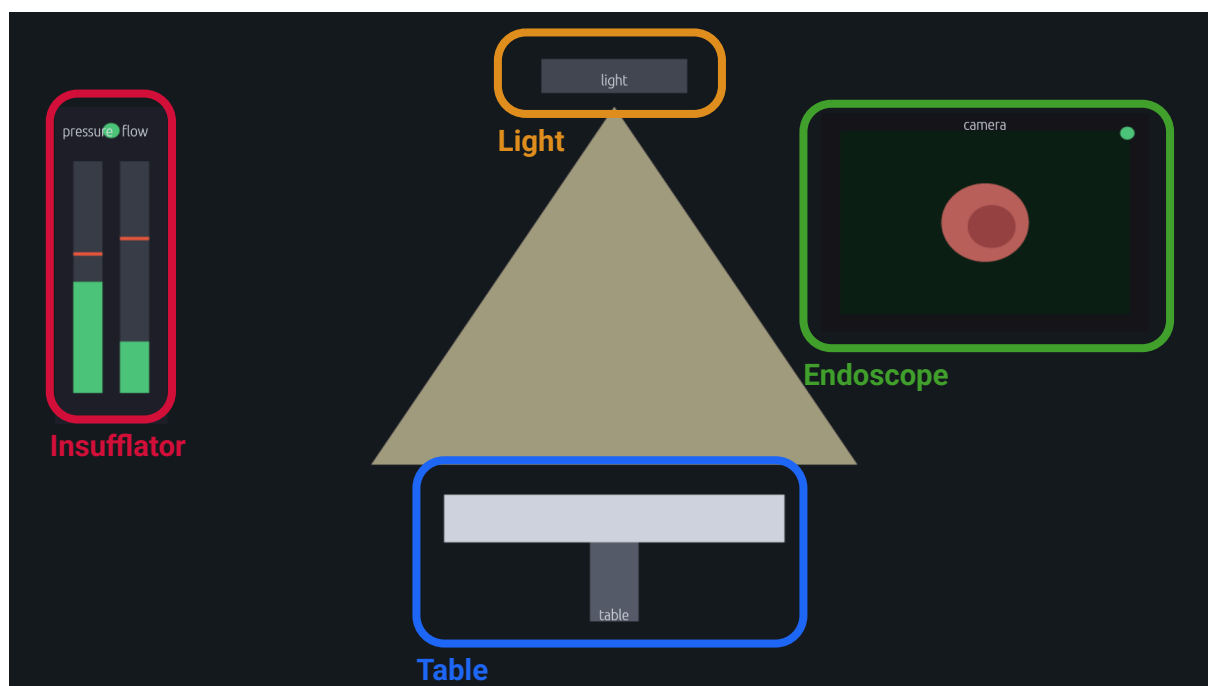


Figure 2.1: The virtual operating room with the controlled equipment highlighted in color: insufflator, light, table, and endoscope.

Table 2.1: Equipment fields represented in the virtual room.

Object	Example fields
Insufflator	<code>gas_flow_l_min</code> , <code>target_pressure_mmhg</code> , <code>insufflator_active</code>
Light	<code>light_brightness</code> , <code>light_mode</code>
Table	<code>table_tilt_degrees</code> , <code>table_height_cm</code>
Endoscope	<code>zoom_level</code> , <code>white_balance_locked</code> , <code>irrigation_active</code>

Table 2.2: System-One question types and example outputs.

Primitive	Question	Example answer space	Example output
Choice	Which team should handle this ticket?	<code>billing</code> , <code>technical</code> , or <code>account</code>	<code>choice: "billing"</code>
Score	How frustrated is this customer?	0 = calm, 1 = frustrated, 2 = very frustrated	<code>score: 1.4</code>
Noul	Does this message request a refund?	True or false	<code>noul: 0.95</code>

There are a lot of complex components involved to make this work reliably in the real world. Luckily, the virtual room simplifies things greatly and allows us to focus on the narrow question of whether a System-One model can turn short utterances into actionable commands we can automatically apply to the objects in the room.

2.2 From utterance to room state

The API gives the model the current state, the utterance, and questions about possible actions to modify the fields/attributes of the objects in the room. Tab. 2.2 shows the three kinds of questions in the API. The possible answers are predetermined by the application, read more on that in .

For example, a question could ask which of several light modes the speaker requested or what brightness value they want. Prog. 2.1 shows a request for “Zoom the camera in.” The response score of 3.79 is not a defined “one step in” command. Application logic still has to map it to a target and enforce allowed bounds.² The example also asks about the light, where `null` means no requested change.

²The JSON example is not representative for the actual application choice schema.

```
1 {
2   "model": "jev-latest",
3   "state": {
4     "utterance": "Zoom the camera in",
5     "camera_zoom": 3
6   },
7   "questions": {
8     "camera_zoom": {
9       "type": "score",
10      "instructions": "Score the target zoom level of the camera",
11      "criteria": [
12        "The lowest zoom level",
13        "Between lowest and medium zoom level",
14        "Moderate zoom level",
15        "Between medium and highest zoom level",
16        "The highest zoom level"
17      ]
18    },
19    "light_mode": {
20      "type": "choice",
21      "instructions": "Choose the target light mode based on the utterance",
22      "criteria": {
23        "normal": "Normal light operating mode",
24        "cavity_focus": "Cavity focus light operating mode",
25        "red_light": "Red light operating mode",
26        "null": "No light change requested in the utterance"
27      }
28    },
29    "light_activated": {
30      "type": "boolean",
31      "instructions": "Should the light be activated or deactivated?"
32    }
33  }
34 }
```

Program 2.1: A System-One model input: an utterance and current room state in JSON.

```
1 {  
2   "model": "jev-latest",  
3   "answers": {  
4     "camera_zoom": {  
5       "type": "score",  
6       "score": 3.79,  
7       "confidence": 0.95,  
8       "probabilities": {  
9         "0": 0.01,  
10        "1": 0.02,  
11        "2": 0.03,  
12        "3": 0.26,  
13        "4": 0.67,  
14        "5": 0.01  
15      }  
16    },  
17    "light_mode": {  
18      "type": "choice",  
19      "choice": "null",  
20      "confidence": 0.95,  
21      "probabilities": {  
22        "normal": 0.04,  
23        "cavity_focus": 0.01,  
24        "red_light": 0.01,  
25        "null": 0.94  
26      }  
27    },  
28    "light_activated": {  
29      "type": "noul",  
30      "noul": 0.921,  
31      "confidence": 0.921  
32    }  
33  }  
34 }
```

Program 2.2: The model's prediction: typed field outputs per equipment, not direct hardware commands.

In Prog. 2.2 the model output is not the room state itself. Application code still has to interpret the answer, apply limits, and decide whether to act based on confidence thresholds. The confidence score of a valid `choice` is very important, as it cannot name an option outside its criteria answer space, but it can still choose an incorrect option. So that is our only way to know how much to trust that confined answer.

Table 2.3: Illustrative candidate criteria for two numeric room fields.

	Base	20	95
<i>light_brightness</i>	<code>null</code>		
	<code>increase:1</code>	<code>set:20</code>	<code>set:95</code>
	<code>decrease:1</code>	<code>increase:20</code>	<code>increase:95</code>
	<code>set:0</code>	<code>decrease:20</code>	<code>decrease:95</code>
	<code>set:100</code>		
<i>table_tilt</i>	<code>null</code>	<code>set:20</code>	<code>set:95</code>
	<code>increase:1</code>	<code>increase:20</code>	<code>increase:95</code>
	<code>decrease:1</code>	<code>decrease:20</code>	<code>decrease:95</code>
	<code>set:0</code>		
:	:	:	:

2.3 Defining the choices

In some cases the answer space is trivial, like for the light mode, which is either `Normal`, `AmbientRed`, `CavityFocus`, or `null` if no change is requested. But what about other continuous fields like the brightness?

Let's take a look at the utterance “Turn up the brightness by *twenty* percent and set the table tilt to *ninety five* degrees.”. Here the highlighted numbers are added as candidates to the criteria table for every actionable field, as shown in Tab. 2.3. By isolating the numeric values, we delegate the task of associating concrete values to actionable deltas to the model. This is the heavy lifting part of the task.

2.4 Running locally

As mentioned in Sec. 1.2, running the model locally is important for GDPR compliance. The less hardware is required, the easier it is to set up for institutions, especially hospitals. So naturally I wanted to try out how far we can get with one.

JevK5 has a model version with 2B parameters that can somehow run on my RTX 2080. The server exposes a System-One-compatible API, so the surrounding request format stays close to the hosted Jev example. I use the model and package revision shown in Prog. 2.3, which is enough to reproduce the environment setup with [uv](#).

toml

```
pyproject.toml
1 [project]
2 name = "jevk5-timerertim"
3 version = "0.1.0"
4 requires-python = "≥3.13"
5 dependencies = [
6     "jevk5[fast]",
7     "causal_conv1d; sys_platform == 'linux'"
8 ]
9
10 [build-system]
11 requires = ["uv_build ≥ 0.12.17, < 0.13.0"]
12 build-backend = "uv_build"
13
14 [tool.uv.sources]
15 causal_conv1d = { index = "astral-gpu-cu130" }
16 jevk5 = { git = "https://github.com/allebee/jevk5", rev = "v0.3.0" }
17
18 [[tool.uv.index]]
19 name = "astral-gpu-cu130"
20 url = "https://wheels.astral.sh/simple/cu130"
21 explicit = true
```

Program 2.3: The pyproject.toml file for the JevK5-2B model used.

I start the server with `uv run jevk5-serve --model alibiserikbay/JevK5-2B`.

3 Evaluation

3.1 Dataset and procedure

I use five virtual surgery scenarios:

- appendectomy,
- hernia repair,
- fundoplication,
- cholecystectomy, and
- sleeve gastrectomy.

The benchmark setup uses 1,000 generated cases with 12 utterances each, including small talk that should not change room state. The benchmark uses `inclusionai/ling-3.0-flash-vl` as the generator for cases and expected output states. That makes a large test set quickly, but generated labels are not expert-reviewed ground truth. You can find the benchmark in the [TimerErTim/stop](#) repository.

I evaluate each utterance on its own and count the predicted field values that match the expected values. The comparison data contains 12 000 Jev entries and 2256 JevK5-2B entries from 188 local-model cases. The paired comparison uses those same entries. JevK5-2B is correct where Jev is not on 4 entries. Jev is correct where JevK5-2B is not on 710.

3.2 Overall results

Table 3.1: Overall metrics for Jev and JevK5-2B benchmark results.

	Jev (TypeSafe AI)	JevK5-2B 
<i>Field Macro-F1</i>	0.886	0.574
<i>Utterance-level Accuracy</i>	0.887	0.573
<i>Hallucination Rate</i>	0.680 %	8.47 %
<i>Latency p50 / p95</i>	260 ms / 358 ms	15 100 ms / 19 300 ms
<i>Calibration ECE</i>	0.0351	0.287

In the recorded runs, Jev performs better than JevK5-2B on utterance-level accuracy. Jev also performs better in the rolling evaluation, where predictions are applied through an utterance sequence and earlier mistakes can affect later state. I keep these measurements separate because JevK5-2B has fewer evaluated cases. The paired comparison in Tab. 3.1 uses the same cases for both models and points in the same direction.

For the full comparison, the field accuracy is 88.7 % for Jev and 57.3 % for JevK5-2B. In rolling mode, it is 61.9 % and 8.78 %, respectively.

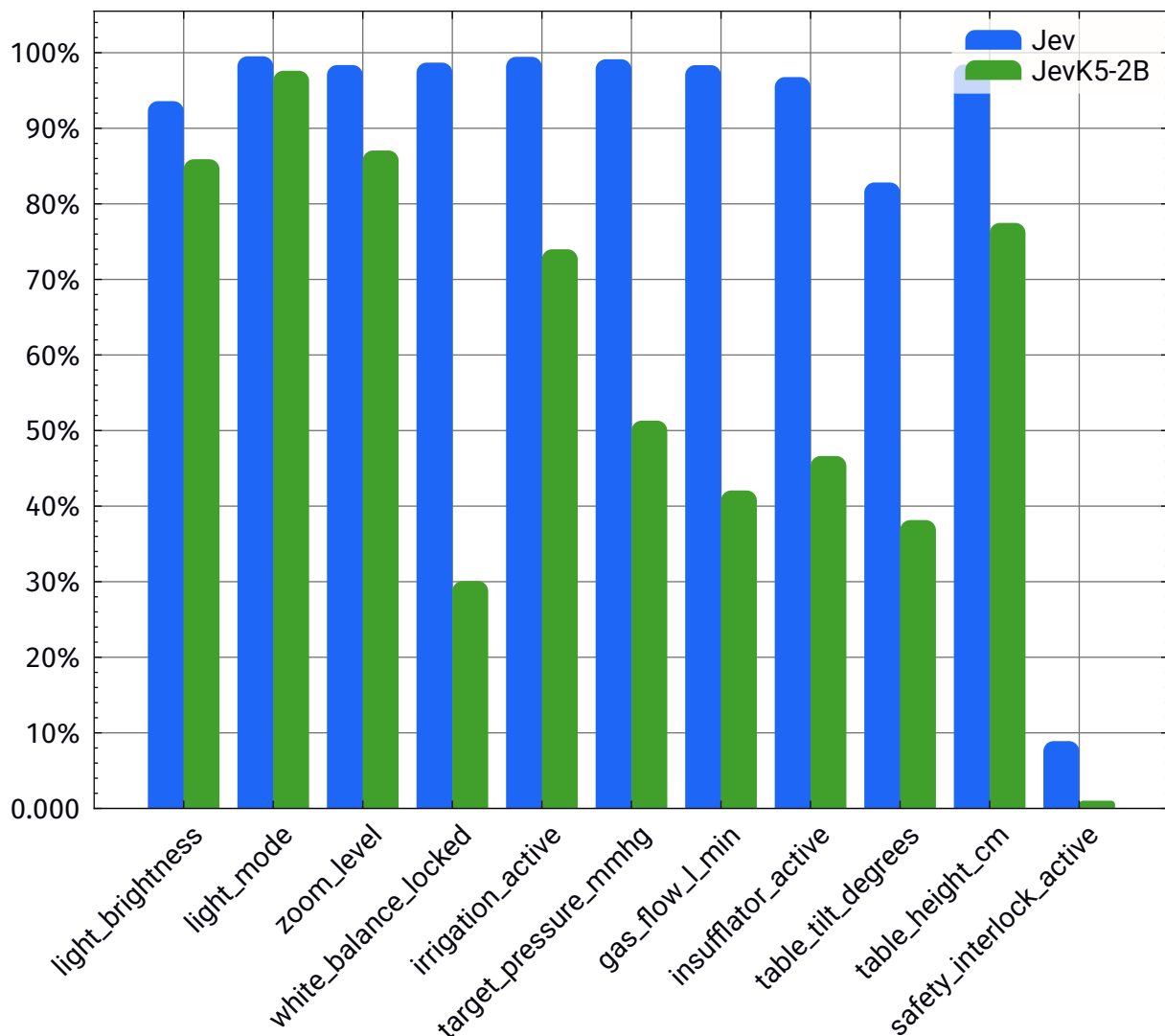


Figure 3.1: Field-level F1 scores from the recorded comparison data. Each bar measures classification quality for one state field.

The table's field macro-F1 averages per-field F1 scores, so a field with many examples does not drown out one with poor performance. The models do not have the same coverage in every sample set, so treat this as a summary of the supplied measurements rather than a like-for-like benchmark.

3.3 It depends on the field

Analyzing Fig. 3.1, Jev handles several fields well in this generated set, but safety interlock activation is a glaring exception: its F1 is 8.91 %, with recall of 4.66 %. JevK5-2B does even worse on that field, with F1 of 1.02 % and recall of 0.513 %. Both models miss nearly all positive interlock cases. For a control system, that alone kills any idea of letting a model own safety decisions. Table tilt is another weaker field for Jev, at F1 82.8 %.

There may be a dataset issue here too. I discovered cases where an utterance like “tilt table by 5 degrees” has an expected state of `table_tilt: -5`, which goes against my intuition and also against Jev’s as it seems.

3.4 Fast enough, on which machine?

Jev’s recorded p50/p95 latency is 261 ms / 359 ms per utterance. That is more than enough to count as real-time interaction. The average human response time is around 200ms, which is excluding the time to physically execute the command, so Jev does not lag behind.

“It runs locally” and “it is useful locally” are two different pairs of shoes.

JevK5-2B’s latency is **15 200 ms / 19 600 ms**. So the local model ran on my RTX 2080, but this particular 2B configuration was not fast enough for the near-real-time interaction I had in mind.

4 Conclusion

System-One's constrained outputs are an appealing interface for equipment-control software. The schema narrows what the model can return, and the API scores each field directly. That is a much better contract than parsing free-form text.

JevK5-2B did not turn out to be a practical local substitute for Jev in this setup. It was less accurate and much slower, and both models missed nearly every positive safety-interlock case. If privacy is the reason to run locally, it currently costs more accuracy and latency than I would accept.

So no, local System-One control is not ready. But the failure modes I saw could be explained by sub-optimal data and instruction setup. Next steps are to investigate whether better data and more guardrails can close the gap without moving the safety boundary.

4.1 Potential improvements

- **Better data, better training.** JevK5-2B runs the default checkpoint. I did not tune it, I just pointed the server at it. My bet is that fine-tuning on messier utterances, ambiguous phrasing, field boundary cases and background noise buys more than swapping architectures. That needs human-reviewed labels instead of the generated ones I used, which would also make the benchmark mean more than "it works on my generated test set".
- **A faster box.** The RTX 2080 is what I had, not what I would buy for this. A newer card, or a smaller or quantized checkpoint, would probably bring latency into a comfortable range. Whether that gets below the ~200 ms human response time on hardware a hospital would actually buy, I have not tested.
- **Better guardrails and guidance.** Rule-based field validation and abstention on low confidence were cheap to add, and they catch the errors the schema lets through on its own. That being said, corrective measures could make a lot of sense in addition to those.

References

1. PEKO, Tim. *TBH*. Online. 23 October 2026. Available from: <https://www.youtube.com/watch?v=dQw4w9WgXcQ>
2. KAHNEMAN, Daniel. *Thinking, Fast and Slow*. First paperback. New York : Farrar, Straus, Giroux, 2011. ISBN 0374533555 9780374533557. In this work the author, a recipient of the Nobel Prize in Economic Sciences for his seminal work in psychology that challenged the rational model of judgment and decision making, has brought together his many years of research and thinking in one book. He explains the two systems that drive the way we think. System 1 is fast, intuitive, and emotional; System 2 is slower, more deliberative, and more logical. He exposes the extraordinary capabilities, and also the faults and biases, of fast thinking, and reveals the pervasive influence of intuitive impressions on our thoughts and behavior. He reveals where we can and cannot trust our intuitions and how we can tap into the benefits of slow thinking. He offers practical and enlightening insights into how choices are made in both our business and our personal lives, and how we can use different techniques to guard against the mental glitches that often get us into trouble. This author's work has transformed cognitive psychology and launched the new fields of behavioral economics and happiness studies. In this book, he takes us on a tour of the mind and explains the two systems that drive the way we think and the way we make choices.
3. ALMEIDA, Diogo. Introducing System-One Models and Jev. Online. [Accessed 10 October 2026]. Available from: <https://typesafe.ai/blog/introducing-system-one-models-and-jev>
4. Available from: <https://github.com/allebee/jevk5>
5. Available from: <https://github.com/jaredpalmer/kev>
6. Available from: <https://huggingface.co/google/diffusiongemma-26B-A4B-it>
7. Available from: <https://github.com/NandhaKishorM/laya>
8. Available from: <https://docs.astral.sh/uv/>
9. Available from: <https://github.com/TimerErTim/stop>