

Frozen Weights

A brain that changes between fights, models that must be rebuilt

Findings from a recorded tank arena, for autonomy and test-and-evaluation programs

Perslis Research | September 2026

Research prototype · simulation and games · not a certified safety system · nothing fielded

Abstract. Programs that field autonomy face a question most benchmarks do not ask: when the system is beaten, how does it get better, and how quickly can the fix be checked? We report what a recorded BZFlag tank arena shows about it. Five kinds of driver fought in the same free-for-all matches from the same engine state: a rule pilot (ordered rules, no neural network in the loop that decides), three language models (Claude through a command-line interface, DeepSeek, and llama3.2:3b run locally) and BZFlag's built-in AI. In the ten-minute headline match (one match) the rule pilot made 60,140 decisions; the models made 58 to 415, at a median of 1.0 to 7.6 s each, although without time pressure they answered 4 to 6 of 8 exam situations correctly. Across the 20 valid recorded matches the built-in AI led (kills per death 1.84, 45 tank-matches), then the rule pilot (0.61, 43), DeepSeek (0.55), Claude and llama (0.06 each; 7 tank-matches per model). Between matches, and with no weights to change, a learning loop altered two of the rule pilot's ten named switches, each adopted only after a paired test ($z = 3.08$ and 2.68). In equal-number matches against the built-in AI, net kills per tank-minute rose from -1.17 (hand-written pilot, 8 matches) to -0.26 (3 matches; $z = 2.93$ for the difference). It has not caught the built-in AI, which still kills 2.10 per tank-minute to its 1.28. The models' weights were fixed throughout; we did not try to retrain them. We set out implications for test and evaluation, labelled as such, and the limits: small samples, one game, simulation only, nothing fielded.

1 Findings at a glance

Six findings, each with its sample size. The first four are measurements; the fifth is an architectural statement about what could and could not change; the sixth is the list of limits. Implications for programs are in §8 and are labelled as implications, not results. Every number is traced to a file in the companion CLAIMS.md.

1. **Decision rate, not knowledge, is the clearest difference between the rule pilot and the models.** In the same ten-minute match (**one match**), the rule pilot made **60,140** decisions, about 101 a second. DeepSeek made **415** (median 1.01 s each), llama3.2:3b **213** (2.55 s) and Claude **58** (7.55 s, through a command-line interface that adds its own delay). Without time pressure the same models answered 6, 5 and 4 of 8 exam situations correctly (rules 8). A shell in this game expires 3.5 s after it is fired. Speed is necessary but not sufficient: BZFlag's built-in AI decides as often as the rule pilot and beats it.
2. **Across every valid recorded match, the built-in AI led; the rule pilot led the models.** Over 20 matches (2 invalid runs excluded, with reasons): built-in AI 804 kills, 437 deaths (K/D 1.84; 45 tank-matches); rule pilot 394–649 (0.61; 43); DeepSeek 30–55 (0.55), Claude 3–47 (0.06), llama3.2:3b 2–33 (0.06), **7 tank-matches per model**, one of them 0.3 minutes long.
3. **The rule pilot improved between fights, without weights.** A loop that replays every death, changes one named switch at a time and keeps a change only after a paired test promoted two changes in 1 h 43 min of matches, the second 7 h 03 min after the first recorded match. In equal-number matches against the built-in AI, net kills per tank-minute went from -1.17 (hand-written pilot, 8 matches) to -0.43 (1 match) to -0.26 (3 matches). The gain from the hand-written pilot to the current one is $z = 2.93$. A never-firing reference pilot scores -2.05 (1 match).
4. **It has not caught the built-in AI.** The current pilot kills **1.28** per tank-minute to the AI's **2.10** (45 v 74 kills; a real gap, $z = 2.66$). It dies at 1.53 per tank-minute to the AI's 1.73, but 54 v 61 deaths is within chance ($z = 0.65$). After the second promotion the loop tried 25 more decisions, promoted none, and stopped itself when it had no untried change left.
5. **The language models could not have changed between matches in this setup; the rule pilot did.** All three models played every match under one model identifier and one set of settings, with one system prompt and no memory carried from one decision to the next, let alone from one match to the next. Their weights can change only by retraining or fine-tuning, which we did not attempt. The rule pilot was changed by hand between two model matches (v1 to v2) and twice by the loop (v2 to v3 to v4), each change a named, readable, reversible edit. This is a statement about architecture, not a measured comparison of learning rates.
6. **Limits.** Small samples (one headline match; 1 to 8 head-to-head matches per pilot version; 7 tank-matches per model), one game, simulation only, one test machine, Claude reached only through a slow command-line route, engineers wrote the switches the loop chose from, and the built-in AI is still the stronger player. Nothing here has been fielded or replicated by a third party.

What this paper does not claim. It does not claim that rules beat language models in general, that the learned pilot is good at the game (it still loses), that the loop learned during a fight (it learns between matches), or that the language models cannot be improved: they could be retrained, or given memory and examples in their prompt, and we tested neither. The rule pilot beat the models before any automatic learning; the learning was measured against the built-in AI, not against the models.

2 The question for a program office

A test-and-evaluation lead who watches an autonomous system lose an engagement has three practical questions. *Why did it lose? What will change so that it does not lose the same way again? How will anyone know the change helped and hurt nothing else?* For a system whose decisions come from a large trained model, the honest answers today are often: we can replay the inputs but not the reasoning; the change is a new round of training or fine-tuning; and the check is a new evaluation of the whole model, because a weight update is not confined to the behaviour one meant to fix. Networks trained on one task and then another are known to lose earlier skills (“catastrophic forgetting” [14, 13]).

This paper looks at those three questions in a small, fully recorded setting: tank battles in BZFlag, an open-source 3D tank game [6]. It is a companion to the research paper *Rules at the Wheel* [1], which measured a rule pilot against three language-model drivers and the game’s built-in AI, and replayed the pilot’s deaths to find why the built-in AI beats it. That paper froze its data at 01:05:50 on 27 September 2026, while the pilot’s learning loop was still running. This white paper adds what the loop recorded after the freeze, up to its last decision at 11:49:06 that morning, and sets out what the whole record says for a defense reader.

The words we use

Rule pilot. A short, ordered list of rules (dodge, get unstuck, engage, hunt, explore) that turns the game state into one action per frame. No neural network is in the loop that decides. Every decision is logged with the rule that fired and a one-line reason.

Switches (the genome). Ten named settings that change how the rules behave, such as how close a passing shell must be before the pilot dodges it. Every change the loop makes is a change to one switch.

The loop. The pilot’s learning procedure: replay the deaths, name the commonest cause, change the one switch that addresses it, test old against new in the same matches, and keep the change only on strong evidence. It runs *between* matches.

Head-to-head. A match of rule tanks of one version against the same number of built-in AI tanks and nothing else. It is the fair measure against the built-in AI.

Net per tank-minute. A side’s total kills minus deaths, divided by the total minutes its tanks played. Zero means as many kills as deaths.

Frozen weights. A trained model whose parameters do not change while it is in use. All three language models in the arena ran this way; that is how such models are normally deployed.

3 The arena in one page

The full method is in the companion paper [1]; this is what a reader needs to judge the findings.

- **One game, many clients.** Every tank is its own BZFlag client with its own driver, all on one free-for-all server. A small patch to the client hands each tank’s controls to an external process and sends it one state packet per frame (own position, visible enemies, incoming shots, open space). The built-in AI runs BZFlag’s own unmodified code. The server builds a new random layout for every match. Shells fly at 100 m/s and expire after 350 m (3.5 s); tanks drive at 25 m/s.
- **Same facts for every driver.** The rule pilot and all three language models are handed the same packet. The models get one prompt, one output format and one parser, and a servo that carries out their last command until the next one arrives. A failed or slow model call is counted, never covered for: the tank simply keeps its last command. A model is never asked about a state older than 1.5 s.

- **The models.** Claude (claude-opus-5, low effort) through the `claude -p` command-line interface, the only route available on the test machine, which starts afresh for every decision; DeepSeek (deepseek-chat) through its web API; llama3.2:3b, a model of about three billion parameters, run locally through Ollama. Settings were the same in every match (Table 1).
- **Everything recorded.** Each match writes a video, a scoreboard every 10 s, final results and a black box per tank: every model question and answer with its timing, every rule decision with its reason, every shot, and for later matches the last two seconds before every death. Numbers here come from the JSON records only.
- **Counting.** A run with an `INVALID.md` file (the run’s premise broke) is excluded, with its reason printed (Table 3). The companion paper also applies a stricter rule (at least 1.5 minutes played); we report both.
- **Hardware.** One Apple M4 laptop with 16 GB ran the server, every client, the rule pilot and llama3.2:3b. DeepSeek and Claude answered over the network.

tank	driver string recorded in every match it played
Claude	claude-opus-5 (cli, effort low)
DeepSeek	deepseek-chat (api, json mode, temp 0)
llama3.2:3b	llama3.2:3b (local ollama, schema format, temp 0)

Table 1. One driver string per model across all its matches (7 valid, 2 invalid). No model was fine-tuned, retrained or given memory of earlier matches.

4 Finding 1: decision rate, not knowledge

The headline match (20260926-183228) put one tank of each kind in the same ten-minute free-for-all (9.92 minutes played). It is **one match**, and we use it only for what one match can show: how often each driver decided.

driver	decisions	median time per decision	kills–deaths	exam (8 situations, no clock)
rule pilot (v1)	60,140	≈0.01 s (every frame)	20–20	8/8
DeepSeek	415	1.01 s	10–18	5/8
llama3.2:3b	213	2.55 s	0–10	4/8
Claude (CLI)	58	7.55 s	1–15	6/8
BZFlag built-in AI	every frame	in the client	38–6	–

Table 2. The headline match (one match) and the written exam. Median time includes failed calls; the average gap between model decisions, including waits, was 1.4 s (DeepSeek), 2.8 s (llama) and 10.3 s (Claude). The exam gave each driver eight hand-built situations with no time limit; Claude’s exam answers took a median 24 s, DeepSeek’s 6.1 s, llama’s 1.4 s. llama3.2:3b gave nearly the same answer to every item, so its 4/8 is not evidence of understanding [1].

What the numbers mean in the fight. A shell here flies 100 m in one second and expires after 3.5 s. In Claude’s median decision time of 7.55 s a shell fired when the question was asked has flown its full range and gone, twice over, and a tank at full speed has moved 189 m. DeepSeek’s 1.01 s is 101 m of shell flight. A model at those rates cannot react to a particular shell; it can only set a course and hope the course is right. The exam shows that the answers were often there (Claude 6 of 8, DeepSeek 5 of 8); what was missing was time.

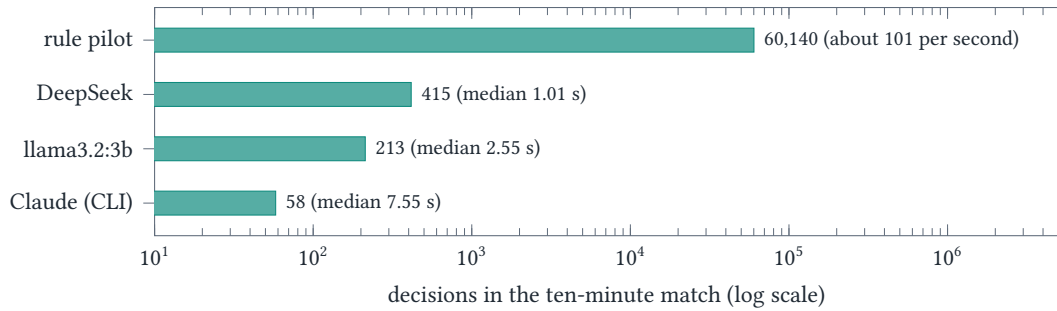


Figure 1. Decisions per driver in the same ten-minute match (one match). The rule pilot decided 145 times as often as DeepSeek and about 1,000 times as often as Claude through its command-line interface.

Two cautions. First, Claude’s time is a property of the transport as much as of the model: the command-line interface starts a new process for every decision, and no faster Claude number is claimed [1]. Second, speed is not enough. The built-in AI also decides every frame, and it beats the rule pilot in every equal match; it wins on better rules, not on more decisions.

5 Finding 2: outcomes across every valid recorded match

The lane’s repository holds 22 recorded runs from the evening of 26 September. Two carry an `INVALID.md`: in 182000 the Claude client left about 10 s into the match and its driver kept deciding on a frozen state; in 192915 training of the image check inside the console starved the controllers. Both are excluded. Table 3 pools the other 20 (109.8 minutes played): 7 matches with the language models and 13 with only the rule pilot and the built-in AI.

driver	all 20 valid runs			16 runs of ≥ 1.5 min	
	tank-matches	kills–deaths	K/D	kills–deaths	kills per tank-min
BZFlag built-in AI	45	804–437	1.84	780–426	3.05
rule pilot	43	394–649	0.61	386–631	1.51
DeepSeek	7	30–55	0.55	29–55	0.90
Claude (CLI)	7	3–47	0.06	3–47	0.09
llama3.2:3b	7	2–33	0.06	2–32	0.06

Table 3. Pooled outcomes. A tank-match is one tank in one match. Left: every committed run without an `INVALID.md` (20 runs). Right: the lane’s stricter counting rule, which also drops four runs shorter than 1.5 minutes (191409, 0.3 min, the seventh model match; 214830, 221854, 224233, 0.8 min each). The ordering is the same under both. The companion paper’s larger pool, 38 counted matches from the offload drive up to its freeze, gives built-in AI 1.75 and rule pilot 0.61 [1].

Reading the table. The built-in AI is the strongest driver measured, by a wide margin, in every pooling. The rule pilot has a higher K/D than every model in 5 of the 6 counted model matches and in the pool [1]. DeepSeek, the fastest model (median 1.0 s), is the closest to the rule pilot. Claude and llama3.2:3b made almost no kills. With 6 or 7 matches per model, these are small samples; the direction is consistent, the precise ratios are not.

Which pilot the models met. The rule pilot in the model matches was v1 (first four) and v2 (last three), both written by hand. The automatically learned versions v3 and v4 never played the language models. Whatever the rule pilot’s advantage over the models is, it is not the product of the learning loop.

6 Findings 3 and 4: learning between fights, and where it stopped

6.1 How the pilot learns

The loop follows the fail-first cycle described in an earlier Perslis paper [2]: *fail, observe, explain, build a rule, verify, retry*. In this lane it runs as follows [1].

1. **Observe.** Every death is recorded with the last two seconds of state packets before it.
2. **Explain.** Each fatal shell is replayed along its path, and every candidate dodge rule is re-run on every packet, to give the death one cause (for example “shot up close”, “dodge abandoned”). On the offence side, the pilot’s own shots are graded the same way (for example “firing from too far”).
3. **Build.** The commonest cause not yet tried names one switch to change. The candidate differs from the current pilot in exactly that switch.
4. **Verify.** Two tanks run the current pilot, two the candidate and two the built-in AI, in the same matches, so both face the same map, moment and enemies. The score of each arm is net kills per tank-minute, pooled over matches.
5. **Decide.** Promote only at $z \geq 2$; reject at $z \leq -1$; anything between is “not proven”, the current pilot stays, and the change may return with its earlier matches pooled, up to 9 matches in all.

A promoted change becomes the pilot of every later match. The switches are the only thing the loop can change: a genome with an unknown switch or a bad value is refused, and a unit test pins that. The loop runs between matches; a pilot never changes during a match.

6.2 The record, 23:31 on 26 September to 11:49 on 27 September

The loop’s first A/B match started at 23:31:35. It made 34 decisions up to 11:49:06: **2 promoted, 8 rejected, 24 not proven**, over 83 A/B matches (323 minutes played). The two promotions came in the first 20 A/B matches (77.5 minutes played):

when	change	what it does	paired test
00:27:51	v2 → v3: dodge_miss_m 4.0 → 6.0	dodge any shell whose path passes within 6 m, not 4 m	$z = 3.08$, 2 matches
01:14:10	v3 → v4: juke on → off	after firing, stay on the target instead of swinging 60° off its line	$z = 2.68$, 2 matches

Table 4. The two promotions. v4 differs from the hand-written v2 in 2 of its 10 switches; everything else about the pilot is unchanged. In the promoting matches the candidate went 32–24 against the current pilot’s 17–42 (v3), and 30–18 against 13–26 (v4).

From the first recorded match (18:11:33) to the second promotion (01:14:10) is **7 hours 3 minutes**; that span includes the hand-written versions v1 and v2, built by engineers from watching the recordings, and the construction of the loop itself (committed at 23:22). The automatic part, from the first A/B match to v4, took **1 hour 43 minutes**.

6.3 The measure that counts: head-to-head against the built-in AI

The A/B matches decide promotions, but they put four rule tanks against two built-in AI tanks, so they do not measure the pilot against the AI on equal terms. The lane measures that separately, in head-to-head matches of three rule tanks of one version against three built-in AI tanks and nothing else (Table 5, Figure 2).

Three things in this table matter for a reader deciding how much to trust it. **The gain is measured on different matches from the ones that chose it:** the promotions were decided in A/B matches; the

pilot	matches	ours k-d	net / tank-min ($\pm$ se)	kills / tank-min	deaths / tank-min	built-in AI k-d
v0 (floor, never fires)	1	0-24	-2.05 ± 0.42	0.00	2.05	37-13
v2 (hand-written start)	8	301-554	-1.17 ± 0.14	1.40	2.57	645-389
v3 (learned)	1	15-20	-0.43 ± 0.51	1.28	1.71	28-22
v4 (learned, current)	3	45-54	-0.26 ± 0.28	1.28	1.53	74-61
built-in AI, v4 matches	3	74-61	+0.37	2.10	1.73	-

Table 5. Head-to-head ladder (three rule tanks of one version v three built-in AI tanks). Standard errors use the lane’s Poisson form $\sqrt{k+d}$ / tank-minutes. v0 is a reference floor that drives but never aims, fires or dodges; it was played at 02:13, after v4, to mark the bottom of the scale, and is not the pilot the loop started from. v3 and v0 have one match each. These are the figures published on the Perslis Defense pages [5], recomputed here from the match records.

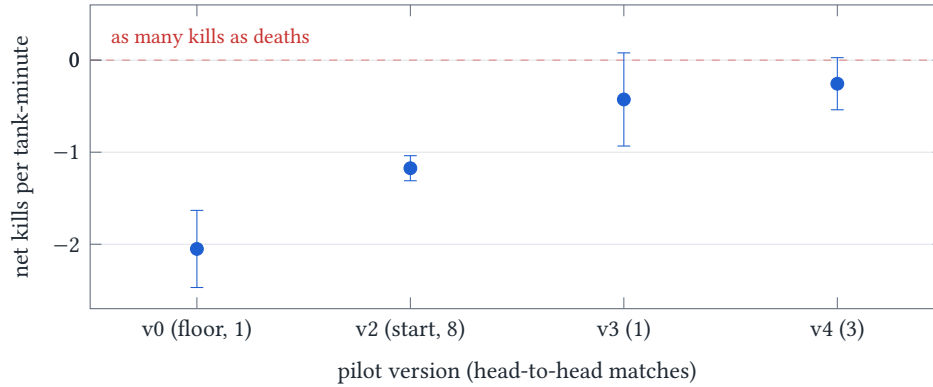


Figure 2. The ladder against the built-in AI, with ± 1 standard error. Improvement from v2 to v4: +0.92 per tank-minute, $z = 2.93$, on head-to-head matches separate from the A/B matches that promoted the changes. v4 remains below zero: it still loses to the built-in AI.

ladder comes from head-to-head matches, so the $z = 2.93$ for v2 to v4 is not the same evidence counted twice. **The sample is small:** v4 has three matches (35 tank-minutes), v3 one. **The pilot still loses:** v4 kills 1.28 per tank-minute to the AI’s 2.10 in the same matches (45 v 74 kills, $z = 2.66$). It dies less often than the AI, 1.53 against 1.73 per tank-minute, but 54 against 61 deaths is within chance ($z = 0.65$); we do not claim it.

6.4 Where it stopped

After the second promotion the loop kept working for another ten and a half hours, with pauses (stop and hand-over requests at 00:44, 01:05, 01:51, 02:12 and 02:29; a full disk stopped it at 02:54; it was restarted at 05:19 and 10:09). It made 25 more decisions: **none promoted**, 7 rejected, 18 not proven (Figure 3). At 11:49:06 it stopped itself with the line “no untried change left for these causes”. Its last explanation of v4’s record: 82% of 653 replayed deaths were shells seen too late to dodge (“shot up close”), and 73% of 5,689 shots were fired from more than 100 m, which the built-in AI dodges. Changes aimed at the second cause (fire only within 100 m or within 50 m) were rejected.

What the engineers did. The loop chose among switches that engineers wrote, some of them during the same night: the dodge-corridor and back-off switches existed when the loop started (23:22), the switch that became v4 (juke) was added at 00:54, twenty minutes before it was promoted, and the firing-range switch at 02:21. The loop picks, tests and keeps or discards; it does not invent new behaviour. That is a limit, and it is also the property a program office may want: the space of changes is written down in

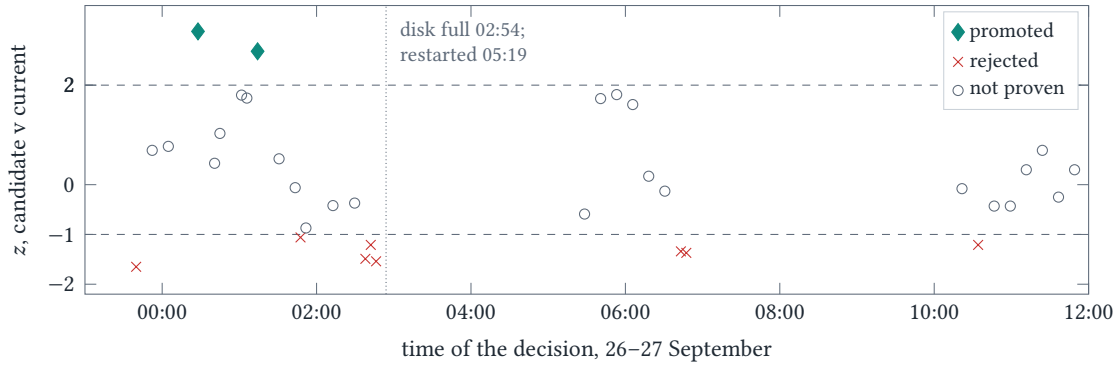


Figure 3. All 34 decisions of the loop. Promote at $z \geq 2$ (upper dashed line), reject at $z \leq -1$ (lower). A change that stays between the lines is “not proven” and may return with more matches (the vertical runs of circles). Both promotions came before 01:15; nothing after reached the line.

advance and can be reviewed before any of it is used.

7 Finding 5: the frozen-weights argument

This section is an argument about architecture, grounded in what did and did not change in the record. It is not a measured comparison of how fast different systems learn: we did not try to improve the language models between matches.

7.1 Three places a decision-maker can change

Any system that picks actions can be changed in one of three places.

1. **Its weights**, the trained parameters of a model. For a language model this means fine-tuning or retraining: collect data, train, and then re-evaluate the whole model, because a weight update is not confined to the behaviour it was meant to fix [14, 13].
2. **Its inputs**, what the model is shown: its prompt, examples, retrieved documents, or a memory of past episodes. Language models can adapt through their inputs without any weight change [9], and agents built this way can improve across episodes by writing reflections or skill libraries into their context [17, 18].
3. **Its decision procedure**, the explicit rules and settings that turn a situation into an action.

7.2 What the record supports

In this arena the language models played every match exactly as they started it. That is not a failing of the models; it is how such models are deployed, and it is what “frozen weights” means. A model that loses a match in the morning plays the afternoon match with the same weights, unless someone retrains it or changes what it is shown.

The rule pilot, by contrast, had its recorded losses turned into named candidate changes. Twice in one night a change survived a paired test and became the pilot of every later match; 32 other decisions left the pilot as it was. Each promoted change is one line (dodge_miss_m: 6.0; juke: false), with a stated reason in the record (“off target after firing, 18% of engaged time”), the matches that tested it, and the score it earned. The whole difference between the current pilot and the hand-written one fits in Table 4.

Two parts of this deserve the most weight for a defense reader. First, **the change is small and legible**. A reviewer can read the diff, see the evidence that admitted it, and put it back. Second, **the check is part of the change**. Nothing entered the pilot without a paired test in the same matches as the version it

	language-model drivers in this arena	rule pilot in this arena
weights	fixed; one model identifier per driver in every match; never retrained	none
inputs	fixed; one system prompt, the current situation only, no memory from one call to the next	the same state packet as every driver
decision procedure	inside the model; not inspectable	ordered rules and 10 named switches; changed by hand once ($v1 \rightarrow v2$) and by the loop twice ($v2 \rightarrow v3 \rightarrow v4$)
what a change looks like	a new model or a new prompt; neither was tried	one line in one switch, with the replays that motivated it and the paired test that admitted it
how it is undone	redploy the old model or prompt	set the switch back

Table 6. What could change between matches, and what did.

replaced, and most candidates did not pass.

7.3 What the record does not support

- It does not show that a learning rule pilot beats a learning language-model system. The models here had no path to learn, by our choice; systems that give a model memory or reflection between episodes exist [17, 18], and we did not build or test one.
- It does not show that learning explains the rule pilot’s advantage over the models. The pilot beat the models as $v1$ and $v2$, before the loop existed.
- It does not show that learning between fights is safe in general. It shows one loop that could change only ten pre-written switches, in a simulator, with an unchanged opponent.

7.4 The same idea elsewhere

Keeping a small, inspectable component that decides, or that bounds what a complex component may do, is an old idea in safety engineering: the Simplex architecture pairs a high-performance controller with a simple verified one [16], and shielding filters a learned policy’s actions against a specification [7]. Earlier Perslis papers apply it to commanded runtimes (orders that only narrow what a pilot may do) [3] and to the fail-first loop [2]. What this record adds is the *learning* half: the small component is also the one that changes, and every change carries its own evidence.

8 Implications for defense programs

Implications, not results

Everything in this section is our reading of what the findings suggest for autonomy and test-and-evaluation programs. None of it has been tested outside a video game, and none of it is a claim that the system described here is ready for any operational use.

I1. Ask how the system changes after a loss, and what evidence admits the change. A program can ask any autonomy vendor three questions the arena makes concrete: where does a change live (weights, inputs or decision procedure); how large is the smallest change; and what paired evidence must a change show before it is used. In this record the answers were: a named switch; one line; $z \geq 2$ in the same matches as the version it replaced, with 32 of 34 decisions declining to change anything.

I2. Learning between engagements is plausible; learning during them is a different claim. The loop needed 20 A/B matches (77.5 minutes of play) for its two promotions, and 83 matches (323 minutes) before it ran out of candidates. That fits a range, a simulator, or the time between sorties, not the middle of an engagement. It also assumed an opponent that did not change in response; a thinking adversary would.

I3. Treat decision latency as a requirement, stated against the threat’s timeline. In this game a shell is gone 3.5 s after it is fired, and the models’ median decision times ranged from 1.0 to 7.6 s. Whatever the domain, a requirement of the form “the decision loop closes within X against threats that resolve in Y” separates architectures before any accuracy test is run. Knowledge measured without a clock (our exam) does not transfer to a fight that has one.

I4. Replayable decisions make a loss explainable. Every rule decision here is logged with the rule that fired and its reason, and every death with the two seconds before it. That is what let the companion paper show that its first explanation of the losses was wrong [1]. The DoD’s AI principles ask for systems that are *traceable* and *governable* [11]; a decision record that names its rule is one concrete form of the first.

I5. Learning should not move the boundary of human authority. In this lane, orders from a person (hold fire, hold position, a target) narrow what the rules may do and are applied in the same decision function whatever the switches are set to; a unit-test suite pins the orders on the pilot, and the switches are a closed list [3]. We did not test every switch against every order, and no head-to-head match ran with orders. The design intent is the one DoD Directive 3000.09 states for autonomous weapon systems: they “will be designed to allow commanders and operators to exercise appropriate levels of human judgment over the use of force” [10].

I6. A change to the algorithm is a review event; make it a small one. The 2023 reissue of DoD Directive 3000.09 calls for senior review again when, among other things, “changes to the system algorithms” substantially differ from those previously approved [12]. A system that learns in the field will meet that clause. A one-line change with its own paired evidence is easier to review than a retrained model, but it is still a change, and a program should assume it triggers review rather than avoids it.

I7. What would have to exist before any of this is fieldable. Independent replication; a domain with physical sensors rather than engine state; an opponent that adapts; a bound on what the switches can do, proven rather than tested; a record signed so that it cannot be edited after the fact; and a test that every order holds under every setting of every switch. None of these exists for this lane today.

9 Limitations

- **Small samples.** The decision-rate finding rests on one ten-minute match. Each language model played 7 valid matches, one of them 0.3 minutes long (6 under the lane’s 1.5-minute rule). The ladder has 1 match for v0, 8 for v2, 1 for v3 and 3 for v4. The z values assume Poisson counts and independent matches; kills and deaths within one match are not independent, so the real uncertainty is larger than the error bars show.
- **Repeated testing.** The loop tested after every match and could promote at any of 34 decisions. Repeated looks at accumulating data raise the chance of a false promotion above what a single $z \geq 2$ test implies [8]. The separate head-to-head ladder (v2 to v4, $z = 2.93$) is the main guard against this; with 3 matches for v4 it is a thin one.

- **Claude through a command-line interface.** No API key was available on the test machine, so Claude was reached only through a CLI that starts afresh for each decision. Its 7.55 s median measures that route as much as the model. No faster Claude figure is claimed, and Claude’s poor score should not be read as a measurement of the model.
- **A small local model.** llama3.2:3b has about three billion parameters and shared the machine with the game; it is not representative of larger local models.
- **No attempt to improve the models.** We did not fine-tune, retrain, prompt-engineer between matches, or give the models memory, examples or reflection. The frozen-weights section is an argument about architecture, not a comparison of learning.
- **Engineers wrote the options.** The loop chose among ten switches, some added by engineers during the same night, and it was paused and restarted by hand. It is automatic selection among written options, not open-ended learning.
- **One game, simulation only.** Every result comes from BZFlag on one Apple M4 laptop, from the engine’s own state rather than sensors, against an opponent (the built-in AI) whose behaviour does not adapt. Nothing transfers to another game, a real vehicle or a real adversary without new evidence.
- **The built-in AI is stronger.** It led every pooling of the record and still out-kills the best learned pilot by a clear margin. The learned pilot’s lower death rate is not significant.
- **Nothing fielded, nothing replicated.** This is a research prototype in a game. It is not a certified safety system, and no third party has repeated these measurements. The records are unsigned JSON files on a local disk.

10 Related work

Our own earlier papers. The arena, the fairness rules, the headline match, the exam and the replay forensics are from *Rules at the Wheel* [1]; this paper cites its measurements rather than repeating them. The learning cycle (fail, observe, explain, build a rule, verify, retry), the paired admission of a change, and credit assignment by replay come from the fail-first paper [2]. Orders that only narrow what a pilot may do, and the commanded-runtime framing, come from the VDSG paper [3]. The observation that language models at the wheel of a driving simulator decide on second-scale latencies is from the CARLA admission paper [4].

Forgetting and retraining. That training a network on new data can destroy what it learned before was shown by McCloskey and Cohen [14]; elastic weight consolidation is one of many remedies [13]. The point we draw from this literature is narrow: a weight change is not local by default, so a retrained model needs re-evaluation as a whole.

Learning without weight updates. Language models adapt to examples placed in their input [9]. Reflexion stores verbal reflections on failures and feeds them back into later attempts [17]; Voyager accumulates a library of code skills while playing Minecraft [18]. These are the fair comparison for the rule pilot’s loop, and we did not run them.

Bounding a complex component. The Simplex architecture keeps a simple, verified controller that can take over from a complex one [16]; shielding filters a learning agent’s actions against a formal specification [7]. Both keep the component that must be trusted small. The NIST AI Risk Management Framework [15] and the DoD’s AI principles [11] set out the governance goals such components serve.

Policy. DoD Directive 3000.09 (2023) requires that autonomous and semi-autonomous weapon systems allow appropriate levels of human judgment over the use of force, and calls for renewed senior review when a system’s algorithms change substantially [10, 12].

11 Conclusion

In a recorded tank arena, three language models played every match exactly as they began it, deciding roughly every 1 to 10 seconds; a rule pilot decided about a hundred times a second and beat them. The built-in AI of the game beat both. Between matches, and with no weights to change, a loop changed two of the rule pilot’s ten switches, each only after a paired test, and, measured on separate matches against the built-in AI, moved net kills per tank-minute from -1.17 to -0.26 , where zero means as many kills as deaths. It did not get past zero, and when it had nothing left to try it stopped.

For a program office the useful part is not the score. It is that the recorded losses produced named candidate changes, each change was one line with its evidence attached, most candidates were refused, and the whole difference between the hand-written pilot and the learned one can be read in a table. Whether that holds outside a video game is the open question, and we have listed what would have to exist before anyone should believe it does.

References

- [1] Perslis Research. Rules at the wheel: tanks, language models and an honest loss. September 2026. research.perslis.com/tank-arena.
- [2] Perslis Research. Fail-first models: failure becomes structure, structure changes the next attempt. September 2026. research.perslis.com/fail-first; summary at www.perslis.com/fail-first-model.
- [3] Perslis Research. VDSG: a commanded admission-control runtime for autonomous agents. September 2026. research.perslis.com/vdsg.
- [4] Perslis Research. Runtime admission control on a photoreal driving simulator. September 2026. research.perslis.com/carla-admission.
- [5] Perslis Defense. Offense and Evolution pages (the BZFlag ladder as published). www.perslis.com/defense/offense; www.perslis.com/defense/evolution.
- [6] BZFlag. Open-source multiplayer 3D tank battle game. github.com/BZFlag-Dev/bzflag.
- [7] M. Alshiekh, R. Bloem, R. Ehlers, B. Könighofer, S. Niekum, U. Topcu. Safe reinforcement learning via shielding. In *Proc. AAAI Conference on Artificial Intelligence* 32(1), 2018. doi:10.1609/aaai.v32i1.11797.
- [8] P. Armitage, C. K. McPherson, B. C. Rowe. Repeated significance tests on accumulating data. *Journal of the Royal Statistical Society, Series A* 132(2):235, 1969. doi:10.2307/2343787.
- [9] T. B. Brown et al. Language models are few-shot learners. *Advances in Neural Information Processing Systems* 33, 2020. arXiv:2005.14165.
- [10] U.S. Department of Defense. DoD Directive 3000.09, *Autonomy in Weapon Systems*. Reissued 25 January 2023.
- [11] U.S. Department of Defense. DOD adopts ethical principles for artificial intelligence (responsible, equitable, traceable, reliable, governable). Press release, 24 February 2020.
- [12] Human Rights Watch. Review of the 2023 US policy on autonomy in weapons systems. 14 February 2023. hrw.org (quotes DoDD 3000.09 §4.1(a)).
- [13] J. Kirkpatrick, R. Pascanu, N. Rabinowitz, J. Veness, et al. Overcoming catastrophic forgetting in neural networks. *Proceedings of the National Academy of Sciences* 114(13):3521–3526, 2017. doi:10.1073/pnas.1611835114.
- [14] M. McCloskey, N. J. Cohen. Catastrophic interference in connectionist networks: the sequential learning problem. *Psychology of Learning and Motivation* 24:109–165, 1989. doi:10.1016/S0079-7421(08)60536-8.
- [15] National Institute of Standards and Technology. Artificial Intelligence Risk Management Framework (AI RMF 1.0). NIST AI 100-1, January 2023. doi:10.6028/NIST.AI.100-1.
- [16] L. Sha. Using simplicity to control complexity. *IEEE Software* 18(4):20–28, 2001. doi:10.1109/MS.2001.936213.
- [17] N. Shinn, F. Cassano, E. Berman, A. Gopinath, K. Narasimhan, S. Yao. Reflexion: language agents with verbal reinforcement learning. *Advances in Neural Information Processing Systems* 36, 2023. arXiv:2303.11366.

- [18] G. Wang, Y. Xie, Y. Jiang, A. Mandlekar, C. Xiao, Y. Zhu, L. Fan, A. Anandkumar. Voyager: an open-ended embodied agent with large language models. arXiv:2305.16291, 2023.

A Where each number comes from

Every number is listed with its source in the accompanying claims file (CLAIMS.md). The sources are the BZFlag lane’s own evidence files, read without modification; no match, game or learning loop was run for this paper. One analysis script, `analysis/fw_analysis.py`, recomputes every derived number. It imports the lane’s own match-counting, head-to-head and pilot code from the commit that recorded the loop’s last decision (b0d4380, 27 September 2026, 15:01), and checks that the lane’s evidence files at that commit are identical to the ones it reads.

result	source
Headline match: decisions, median decision times, scores	<code>evidence/arena/20260926-183228/results.json</code> , <code>meta.json</code> , <code>score-board.jsonl</code>
Written exam	<code>evidence/exam.jsonl</code> (48 answers, six drivers)
Pooled outcomes (Table 3)	<code>results.json</code> of each of the 22 runs in <code>evidence/arena/</code> ; two <code>INVALID.md</code> files; played minutes by the lane’s <code>history.match_row</code>
Head-to-head ladder (Table 5)	the lane’s <code>evolve.benchmarks()</code> over every run on the offload drive (116 run directories, to 20260927-114455)
Loop decisions, promotions, stop (Figure 3)	<code>evidence/pilots/evolution.jsonl</code> (34 records), <code>evolve.log</code> , <code>current.json</code> , <code>progress.json</code>
Genome and its closed list of switches	<code>bzflag_floor/pilot.py</code> at b0d4380; tests <code>test_orders.py</code> and <code>test_unknown_genes_are_refused</code> , run in a scratch copy of the pinned code: 21 passed
When each switch was written	<code>git log -S on pilot.py</code>

Rebuild.

```
python3 -B analysis/fw_analysis.py      # -> analysis/out/fw.json
python3 -B analysis/figure_data.py      # -> tex/figs/*.tex, web-build/figdata.json
cd tex && mkdir -p out && tectonic -X compile paper.tex --outdir out
```