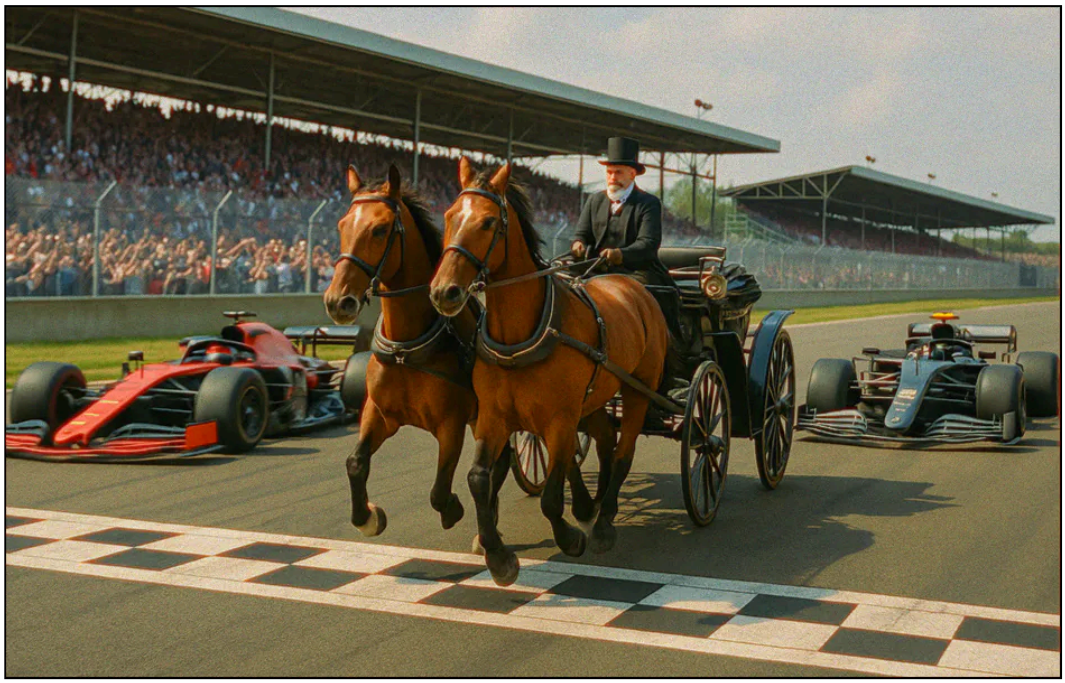
Human Code From 2020 Thrashed Vibe-Coded Agents in Agentic Tests

Originally published at https://www.unite.ai/human-code-from-2020-thrashed-vibe-coded-agents-in-agentic-tests/



ChatGPT and other vibe-coding tools were put to the test in nearly 40,000 matches – and lost to grad student code written before the invention of Large Language Models.

In a new study from the UK, researchers pitted human-coded agents against vibe-coded agents developed with the latest Large Language Models (LLMs), such as ChatGPT-5 and Claude, and found that the agents created without the aid of AI very easily beat the AI-facilitated versions.

Both sets of agents were created by different generations of students from the Artificial Intelligence Laboratory at the Swiss Federal Technology Institute of Lausanne. The non-AI agents were developed as part of coursework in 2020, two years before the inception of ChatGPT and the start of the LLM revolution, while the new agents were created by current students, aided by the latest and best LLMs available.

Even with a rigged game, the vibe-coded solutions could not win, and the top five spots were consistently held by 'raw' agents, with the majority of LLM agents (33 out of 40) beaten effortlessly by 'very simple' baseline agents, across 38,304 challenges in a tournament, across a wide number of variables and circumstances.

The paper states:

> *'Our work demonstrates that while state-of-the-art LLMs can generate code that runs (i.e., free of syntax errors), the generated solution is not competitive to human-designed solutions on dimensions such as strategic planning, optimization, or multi-agent competition.*
>
> *'Thus, this work brings to the forefront this new frontier in code generation, and aims to facilitate the development of benchmarks, datasets, and open-source baselines that stress reasoning-driven code synthesis.'*

The challenge devised was to creatively participate in auctions, across a variety of strategies, and to arrange the logistics of delivering won items to the winners.

The authors note that a number of advantages were given to LLMs, such as intervening in their code to improve their performance – a boon not allowed to the 2020-era code. In spite of this, even when supplied with corrective code that would have definitely improved their outcomes, the LLMs were not able to accept it or use it:

> *'[In] our benchmark, even when we expose a good solution in-context, the LLM is still unable to utilize it.*

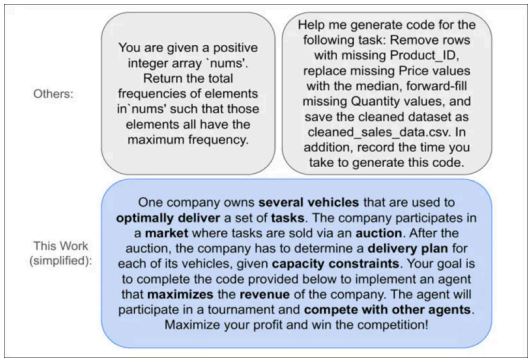
'This result also raises interesting future research questions about the limits of in-context learning and retrieval-augmented problem solving in complex scenarios.'

The LLMs used in the test were GPT-5 Thinking, Gemini 2.5 Pro, Claude Opus 4.1, AND DeepSeek R1*.

The new paper is titled *Can Vibe Coding Beat Graduate CS Students? An LLM vs. Human Coding Tournament on Market-driven Strategic Planning*, and comes from one author at the University of Southampton, and another at the University of Oxford and Alan Turing Institute. The benchmark will, the authors state, be released shortly.

Method

The authors note that traditional tests in this sphere focus on challenges with clearly-defined binary solutions (*correct* or *not correct*), verified through unit tests. Contending that this is not the ideal way to explore the limitations of LLM-aided code, the authors instead devised a more complex challenge scenario, with multiple internal benchmarks and milestones, in which victory is possible, but far from simple:



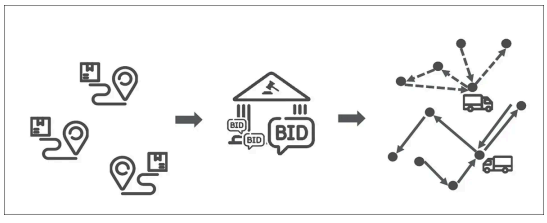
Comparison of standard, unit-test-based approaches (above), and the more open-ended challenge scenario devised by the authors (in blue, below). Source

The Auction, Pickup and Delivery Problem (APDP) used for the authors' study was partly self-selected, because of the availability of a corpus of 2020 student work from the Swiss university; work which sought to create automated agents for the APDP task, prior to any ability to bolster development through AI. Therefore it was relatively easy to task modern students with the same brief, but avail them of current tools.

The authors sought to avoid popular testing frameworks such as HumanEval, BigCodeBench and WebDev Arena (among many others), since this class of testing procedures tends to suffer from data contamination (i.e., instances where the system may have trained on test data instead of respecting a split).

The APDP is a two-stage logistics problem based on reverse auctions and vehicle routing. In the first stage, agents compete to win delivery tasks by submitting bids for how much they should be paid to complete each one. Bidding too high means losing the task; bidding too low can mean losing money.

In the second stage, each agent must create an efficient plan to fulfill only the tasks they won, assigning them to vehicles with different capacities and costs, under time and resource constraints:



In the APDP, companies bid in reverse auctions for delivery tasks, then optimize vehicle routes to fulfill only the tasks they win, aiming to maximize profit.

The goal is not simply to complete the tasks, but to maximize overall profit by anticipating which bundles of tasks will work best together, and predicting the strategies of competitors who are all trying to do the same.

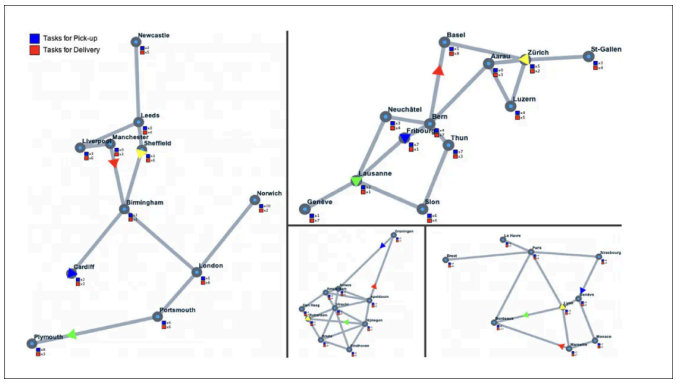
The APDP benchmark raises the difficulty of code generation tasks by introducing strategic planning across a sequence of interdependent auctions, with each bid reshaping the landscape of future choices; and therefore requires agents to reason not just about immediate costs, but about positioning, timing, and long-term consequences.

The core delivery problem is NP-hard, i.e., no algorithm can reliably find the best solution in reasonable time as the number of tasks grows. This makes brute force an unworkable approach, and forces agents to trade precision for speed.

The Race is On

The authors' evaluation compared 40 LLM-coded agents against 17 human-coded agents in a series of head-to-head tournaments. Each of the 12 tournaments used a different combination of four road network topologies, and consisted of all-play-all pairings, with agents facing every other opponent twice: once controlling each of two companies, with different vehicle specs.

This setup yielded 3,192 matches per tournament, totaling 38,304 matches. In each match, 50 delivery tasks were auctioned, defined by their pickup and drop-off points and weight, and drawn randomly across road layouts modeled on Switzerland, France, Great Britain, and the Netherlands:



Simplified road networks used in the tournament: Great Britain (top left), Switzerland (top right), the Netherlands (bottom left), and France (bottom right). Blue and red squares mark pick-up and delivery tasks. Colored triangles show the current positions of agents' vehicles.

Student agents were drawn from a 2020 course tournament. Eight came from the top performers in a single-elimination final, and four more were chosen for strong performance against the baseline agents in head-to-head matches.

The baseline agents followed fixed heuristics. *Naive* calculated total distance and bid accordingly, using only one vehicle and ignoring batching; *ExpCostFixedBid* simulated 10 random tasks, and bid the average marginal cost; *Honest* computed the actual marginal cost of inserting the task into the schedule; *ModelOpponent* did the same but added an estimate of the opponent's cost, bidding the maximum; and *RiskSeeking* blended a time-decaying prior with live cost estimation and opponent modeling – again bidding the higher of the two.

The evaluation included 40 LLM-coded agents built using the (aforementioned) GPT-5 Thinking, Claude Opus 4.1, Gemini 2.5 Pro, and DeepSeek R1. Each model was prompted with five distinct strategies, applied twice per model.

Two strategies used static prompts written by different authors, while a third asked the model to self-reflect and revise its own output; another involved critique and revision by a separate LLM. The final strategy used GPT-4 to synthesize a new prompt by reviewing all four prior approaches.

The base prompt reflected the original student assignment, describing the delivery environment and instructing the model to bid and plan to maximize profit, without relying on high-complexity methods.

All LLM agents were tested in both self-play and tournament settings until all observable bugs were fixed. Bug-fixing was handled autonomously by the LLMs themselves, prompted with the error information.

Common LLM failures, the paper notes, included violations of timeout limits, failure to pick up or deliver assigned tasks, and breaches of vehicle capacity constraints – errors which often arose from disregarding explicit instructions, or from faulty replanning logic[†]:

> *'Another common issue we found (mostly with Gemini, Claude, and DeepSeek, and not so much with GPT) is that quite often the LLM would consistently fail to resolve a bug.*
>
> *'For example, an agent would consistently time-out, despite multiple (e.g., 5 − 15) cycles of prompting the LLM with the error and receiving the updated version of the code.*
>
> *'The only solution we found for such situations (where the LLM repeatedly fails to resolve the exact same bug) is to* re-start from scratch. *Overall, we observed the need for* significant manual effort to achieve bug-free code. *We had to generate substantially more agents to get the 40 bug-free ones we evaluated.'*

The results shown below summarize outcomes from 12 double round-robin tournaments, spanning four network topologies and three tournaments per topology, yielding the best part of 40,000 matches:

Agent	Avg #Wins / Tour	SD #Wins / Tour	Avg #Losses / Tour	SD #Losses / Tour	Total Wins	Total Losses	Winrate
Student 1	108.167	1.193	3.833	1.193	1298	46	0.9658
Student 2	104.917	2.539	7.083	2.539	1259	85	0.9368
Student 3	103.917	2.466	8.083	2.466	1247	97	0.9278
Student 4	103.25	1.815	8.75	1.815	1239	105	0.9219
Student 5	96.5	2.908	15.5	2.908	1158	186	0.8616
LLM(O, IR, 1)	95.417	2.314	16.583	2.314	1145	199	0.8519
LLM(O, A2, 1)	94.583	2.314	17.417	2.314	1135	209	0.8445
Student 6	93.167	1.899	18.833	1.899	1118	226	0.8318
Student 7	93.167	3.563	18.833	3.563	1118	226	0.8318
LLM(O, A1, 1)	86.083	3.029	25.917	3.029	1033	311	0.7686
LLM(O, GEN, 2)	84.083	6.947	27.917	6.947	1009	335	0.7507
LLM(O, CR, 2)	83.5	4.442	28.5	4.442	1002	342	0.7455
Student 8	83.417	4.122	28.583	4.122	1001	343	0.7448
RiskSeeking	82.417	3.343	29.583	3.343	989	355	0.7359
LLM(O, GEN, 1)	80.667	4.355	31.25	4.372	968	375	0.7208
ModelOpponent	80.583	3.26	31.417	3.26	967	377	0.7195
LLM(D, A1, 1)	79.417	3.965	32.583	3.965	953	391	0.7091
ExpCostFixedBid	77.167	4.951	34.833	4.951	926	418	0.689
LLM(O, IR, 2)	73.917	3.502	38	3.618	887	456	0.6605
LLM(O, A1, 2)	72.417	2.193	39.583	2.193	869	475	0.6466
LLM(G, A1, 2)	68.5	3.555	43.5	3.555	822	522	0.6116
LLM(A, GEN, 2)	67.917	2.968	44.083	2.968	815	529	0.6064
LLM(G, IR, 2)	65.917	2.314	46.083	2.314	791	553	0.5885
Student 9	64.167	11.044	47.833	11.044	770	574	0.5729
LLM(G, A1, 1)	64	4.243	47.917	4.316	768	575	0.5719
LLM(G, IR, 1)	60.333	3.725	51.667	3.725	724	620	0.5387
LLM(O, A2, 2)	59.333	4.499	52.667	4.499	712	632	0.5298
LLM(D, CR, 1)	55.083	6.694	56.833	6.59	661	682	0.4922
LLM(G, GEN, 2)	53.167	3.664	58.833	3.664	638	706	0.4747
LLM(D, GEN, 2)	52.083	9.06	59.917	9.06	625	719	0.465
Honest	50.583	3.848	61.417	3.848	607	737	0.4516
Student 10	48.833	2.98	63.167	2.98	586	758	0.436
LLM(D, IR, 1)	48.583	10.211	63.417	10.211	583	761	0.4338
LLM(A, A1, 1)	48	4.69	64	4.69	576	768	0.4286
LLM(G, A2, 1)	47.25	3.864	64.75	3.864	567	777	0.4219
LLM(A, CR, 1)	43.833	4.609	68.167	4.609	526	818	0.3914
LLM(A, A1, 2)	43.75	2.05	68.25	2.05	525	819	0.3906
Student 11	42.083	5.664	69.917	5.664	505	839	0.3757
LLM(A, IR, 1)	39.5	2.541	72.5	2.541	474	870	0.3527
Naive	36.75	1.712	75.25	1.712	441	903	0.3281
Student 12	36.333	1.775	75.667	1.775	436	908	0.3244
LLM(D, A2, 1)	33.917	2.193	78.083	2.193	407	937	0.3028
LLM(A, GEN, 1)	30.167	1.749	81.833	1.749	362	982	0.2693
LLM(D, A2, 2)	29.833	2.038	82.167	2.038	358	986	0.2664
LLM(G, A2, 2)	27	2.256	85	2.256	324	1020	0.2411
LLM(A, A2, 1)	26.333	0.985	85.667	0.985	316	1028	0.2351
LLM(O, CR, 1)	25	3.411	87	3.411	300	1044	0.2232
LLM(A, IR, 2)	24.333	8.542	87.667	8.542	292	1052	0.2173
LLM(A, A2, 2)	24	1.809	88	1.809	288	1056	0.2143
LLM(A, CR, 2)	23.333	1.557	88.667	1.557	280	1064	0.2083
LLM(D, GEN, 1)	22.5	1.784	89.5	1.784	270	1074	0.2009
LLM(D, A1, 2)	13.333	1.826	98.667	1.826	160	1184	0.119
LLM(G, CR, 1)	9.5	1.087	102.5	1.087	114	1230	0.0848
LLM(G, GEN, 1)	9.167	0.937	102.833	0.937	110	1234	0.0818
LLM(D, IR, 2)	7.75	0.622	104.25	0.622	93	1251	0.0692
LLM(G, CR, 2)	7.25	1.422	104.75	1.422	87	1257	0.0647
LLM(D, CR, 2)	5.667	0.985	106.333	0.985	68	1276	0.0506

For context, each agent played 112 matches per tournament, so the maximum possible average for wins or losses per agent is 112. Standard deviation (SD) reflects variability across tournaments. Human-coded agents appear in bold. LLM-coded agents are labeled by model (O = GPT-5 Thinking, G = Gemini 2.5 Pro, A = Claude Opus 4.1, D = DeepSeek R1), followed by a two-letter prompt strategy code and a digit indicating whether the agent is the first or second generated with that prompt. Source

In regard to the results shown above, the authors state[†]:

> *'LLMs did not generate expected/competitive code even in simpler variants of the APDP problem (despite the code being largely syntax-bug-free). This underlines the importance of reasoning-driven code evaluation benchmarks that go beyond auto-complete and identify new weaknesses of LLMs.'*
>
> *'Our results demonstrate a clear superiority of human-coded agents: (i) The* ***top 5 spots are consistently held by student agents****, and (ii)* ***the majority of LLM agents (33 out of 40) are beaten by very simple baseline agents*** *(such as the expected cost fixed bid).*
>
> *'Importantly, we did not debug the student code (while we thoroughly tested/debugged the LLM code, both in self-play and tournament [settings]). Every time a student agent crashed, we automatically gave the win to the LLM. A large number of these crashes would be easy to fix (e.g., agents timed-out), thus student agents could potentially* rank even higher.*'*

As a further experiment, GPT-5 Thinking was prompted to improve the code of the top-performing human agent, *Student 1*; but the now LLM-modified agent subsequently fell to tenth place, now the worst of all the human scores. Instead of enhancing the solution, the LLMs' changes degraded it by nearly 20%.

The authors conclude:

> *'[Our] results highlight important limitations of LLM code generation, most notably their limited reasoning and planning capabilities while generating . Modern LLMs are able to provide syntax-bug-free code that runs, but that is not the benchmark we should be using to measure progress towards advanced general AI.'*

Conclusion

The authors themselves observe toward the close of the paper that vibe-coding has empowered people of all technical backgrounds, and characterize the practice in a positive light, as a leveling force. However they also imply that because vibe-coding has only just arrived, its limits are not known, and may be assumed to be rather higher than one can realistically expect.

They close their offering by calling for a goal-shift '*from code that compiles to code that competes*'.

One question that the casual reader of this interesting new paper may have is whether the authors are punching up or down, since the agentic task in question is considerably more complex and involved than spitting out PowerShell scripts and other forms of minor functionality and fixes for which vibe-coding is well-suited.

* *Please note that the paper refers continuously to 'Deep****Think*** *R1', which appears to be non-existent, turning up only a handful of references on the internet (presumably from other authors that have mis-written 'DeepSeek R1)'. If this is my error, please contact me via my profile details, and I will amend.*

† Authors' emphasis, not mine.

First published Wednesday, November 26, 2025. Amended 17:35 est for formatting.

Writer on machine learning, domain specialist in human image synthesis. Former head of research content at Metaphysic.ai, until its dissolution into DNEG's Brahma.ai.
Portfolio site: martinanderson.ai
Contact: martin@martinanderson.ai

Discover More