Can Vibe Coding Beat Graduate CS Students? An LLM vs. Human Coding Tournament on Market-driven Strategic Planning

Panayiotis Danassis
University of Southampton
Southampton, United Kingdom
p.danassis@soton.ac.uk

Naman Goel
University of Oxford and Alan Turing Institute
Oxford, United Kingdom
naman.goel@alumni.epfl.ch

ABSTRACT

The rapid proliferation of Large Language Models (LLMs) has revolutionized AI-assisted code generation. This rapid development of LLMs has outpaced our ability to properly benchmark them. Prevailing benchmarks emphasize unit-test pass rates and syntactic correctness. Such metrics understate the difficulty of many real-world problems that require *planning, optimization, and strategic interaction*. We introduce a multi-agent reasoning-driven benchmark based on a real-world logistics optimization problem (Auction, Pickup, and Delivery Problem) that couples competitive auctions with capacity-constrained routing. The benchmark requires building agents that can (i) bid strategically under uncertainty and (ii) optimize planners that deliver tasks while maximizing profit. We evaluate *40 LLM-coded* agents (by a wide range of state-of-the-art LLMs under multiple prompting methodologies, including vibe coding) against 17 *human-coded* agents developed *before the advent of LLMs*. Our results over 12 double all-play-all tournaments and $\sim 40k$ *matches* demonstrate (i) a clear superiority of human(graduate students)-coded agents: the *top 5 spots are consistently won by human-coded agents*, (ii) *the majority of LLM-coded agents (33 out of 40) are beaten by very simple baselines*, and (iii) given the best human solution as an input and prompted to improve upon, the best performing LLM makes the solution significantly worse instead of improving it. Our results highlight a gap in LLMs' ability to produce code *that works competitively in the real-world*, and motivate new evaluations that emphasize reasoning-driven code synthesis in real-world scenarios.

KEYWORDS

Large Language Models (LLMs), Code generation, Vibe Coding, Benchmarks, Human Evaluation

1 INTRODUCTION

Large Language Models (LLMs) have demonstrated an impressive ability to generate executable code, free of syntax errors [1, 3, 23]. Software engineers increasingly use LLMs for code generation, bug detection, code refactoring, and more [9, 12, 43]. 'Vibe-coding' has empowered users of all technical backgrounds to turn their ideas into code in seconds [33].[1] As a result, there is significant interest in evaluating and improving the coding capabilities of LLMs. Recent literature has proposed a plethora of benchmarks evaluating various aspects of code generation such as functional correctness, reliability, robustness, execution time, code security, etc. [26, 42, 49]. Most existing benchmarks are composed of problems whose solutions can be easily verified by running unit tests (e.g., see examples in Figure 1 (top)). Achieving state-of-art performance on these benchmarks is indeed an impressive milestone for AI. Yet, if we look *beyond autocomplete*, real-world software development is far more complex. We argue that it is time to expand the frontier in code generation even further by asking:

> *Does high performance on existing coding benchmarks translate to the ability to solve real-world coding problems, ones requiring multi-agent strategic interaction, planning, optimization, and advanced algorithm design that can be highly non-trivial even for experienced human software engineers?*

We introduce a real-world optimization benchmark for LLM code generation. The proposed benchmark requires LLMs to complete code that optimizes the operations of a logistics company, specifically the pickup and delivery of parcels (see Figure 2), with the ultimate goal of maximizing profit. The LLMs need to implement an agent that competes against other agents, optimally bids for tasks (tasks are assigned via a reverse first-price sealed-bid auction), and optimally plans for the pickup and delivery of the won tasks. As

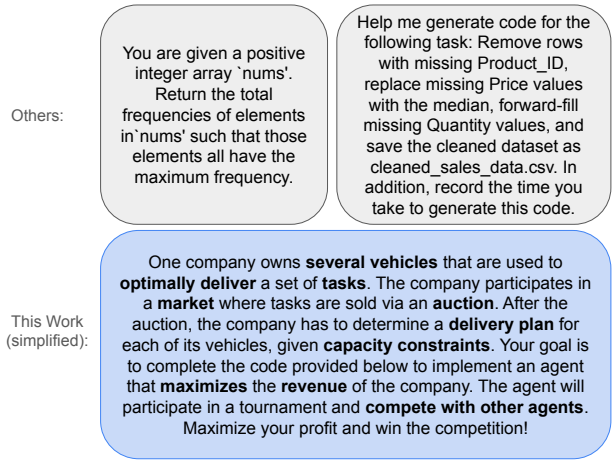

Figure 1: Traditional benchmarks (top) focus on problems with clearly defined correct or incorrect solutions, typically verified through unit tests. In contrast, our benchmark (bottom) involves complex tasks such as planning, constraint optimization, modeling competitors, competitive strategy design, and advanced algorithm development — challenges that remain highly non-trivial even for experienced software engineers. Top from [27] (left) and [26] (right).

[1]The term 'vibe-coding' colloquially refers to a software development approach that uses LLMs (general-purpose or specialized for coding) to generate code from natural language prompts.

Part of this work was conducted while P.D. was with Telenor Research.
Code available at: https://panayiotisd.github.io/apdp_bench/.

such, the problem incorporates challenges from the domains of *multi-agent systems (MAS)*, *auctions*, and *constraint optimization*.

Devising a suitable benchmark is one part of the research challenge addressed in this work; the second relates to evaluation metrics. The Auction, Pickup, and Delivery Problem (APDP) is an open ended problem, one that does not admit a closed-form solution (due to real-time constraints for bidding and planning, bounded rationality, etc.). Thus, pass/fail designations of the generated code are not feasible, nor can we estimate how close we are to the optimal solution. Instead, we compare against a wide range of human-coded agents, developed *before the advent of LLMs* (i.e., without any AI-assisted programming). The APDP was given as an assignment in the postgraduate Intelligent Agents course at EPFL. Students had two to three weeks to develop an agent, which then competed in a single-elimination tournament for extra course credits. We selected 12 student agents from the class of 2020 and 5 baseline agents developed by members of the Artificial Intelligence Laboratory at EPFL.

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.*

Summary of our Contributions

(1) **We provide a novel, reasoning-driven benchmark for automatic code generation.** The Auction, Pickup, and Delivery Problem (Figure 2) in the proposed benchmark combines challenges from the domains of competitive multi-agent systems, auctions, and constraint optimization and requires capabilities to design and code advanced planning and optimization algorithms. The APDP benchmark will be open-sourced.

(2) This is the *first* work, to the best of our knowledge, that answers two crucial questions: **How well does LLM code generation and vibe coding perform beyond autocomplete, in real-world planning settings?** *Are Large Language Models graduate-level coders?* We evaluated a range of state-of-the-art LLMs (both paid and free) from various companies, ones that most people would have access to (GPT-5 Thinking, Gemini 2.5 Pro, Claude Opus 4.1, DeepThink R1), against 17 human-coded agents, developed *before the advent of LLMs*, including 12 agents developed by *students*.

(3) Our results over 12 double all-play-all tournaments and almost *40k matches* (Table 1) demonstrate a clear superiority of student-coded agents. (i) The **top 5 spots are consistently won 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). Perhaps surprisingly, we also find that **LLMs degrade the performance of top human-coded solutions**. When the best performing LLM was given as input the winning student solution and asked to improve it, the resulting agent performed *significantly worse*, dropping to 10th place.

2 RELATED WORK & DISCUSSION

There is much literature on evaluating the coding capabilities of LLMs. Claims about LLM capabilities range from LLMs being excellent coding assistants (e.g., by autocompleting lines of code in an IDE) [16, 47] and LLMs being better than humans in competitive coding [10, 30] to LLMs being able to self-improve their own code [36]. Fully automating software engineering is also considered by many as a step towards "AGI (Artificial General Intelligence)" [2].

There exist several evaluation methodologies for coding LLMs. One way to measure the utility of LLM-produced code is to measure the impact on developers' productivity [7]. A related method is to evaluate the quality of code based on subjective ratings or acceptance rate of code suggestions by humans [12, 35, 44]. While capabilities are continuously evolving with new models being released, prior work have identified various strengths and weaknesses of different models [6, 26, 31]. For example, [7] found that while developers thought they were 20% faster with AI tools, they were actually 19% slower when they had access to AI than when they didn't. Their results (and disagreement with previous studies [16, 47]) raised important questions, e.g., what are the right metrics to evaluate the utility of LLMs in coding tasks, in what ways are coding-LLMs actually useful and where do they still lack capability?

While insightful, such user-centric evaluation methods are also costly and results are difficult to reproduce. Therefore, it is more common for evaluation methods to focus on static benchmarks. Existing benchmarks primarily evaluate two things: functional correctness [13] (with metrics such as pass@k [13], pass-ratio@n [46], CodeBLEU [39], etc.) and efficiency (execution time) [21, 38] of the generated code. Other criteria such as code security, semantic correctness, code interpretability have also drawn attention [26, 42, 49]. Some representative and most commonly used [1, 3, 23] benchmarks are: HumanEval [13], APPS [25], MBBP [6], BigCodeBench[50], LiveCodeBench [27, 45], SWEBench and its variants [15, 18, 28, 34, 48], Aider Polyglot[22], WebDev Arena [14, 44], etc. We direct the interested readers to [20] for a survey of various coding benchmarks. In addition, benchmarks that contain questions, unit tests, and human data from platforms such as Codeforces[2] are often used to compare LLM performance with human coders [10, 30]. While all these benchmarks offer a scalable quantitative way to evaluate models (and contain challenging coding problems), they come with a different set of limitations: data contamination (where models train on test data) [41], limited scope that doesn't reflect real-world and open-ended tasks, lack of adaptability and creative testing, etc.

Therefore, there is a growing interest in the community to find novel and more robust ways to test the models. For example, [24] design and evaluate a code generation system for data science programming problems in Kaggle competitions. While data science solutions (e.g., ML classification models) are evaluated on hidden test sets, they are also different from other coding competition problems in the sense that (i) they are often closer to real-world scenarios and (ii) evaluation metrics are usually non-binary (i.e., even a "correct" solution can be improved through, e.g., model selection, feature engineering, hyper-parameter tuning, etc.).

When it comes to real-world competitions, in July 2025, at the "AtCoder World Tour Finals 2025 Heuristic", an OpenAI system

[2] https://codeforces.com/blog/entry/68288

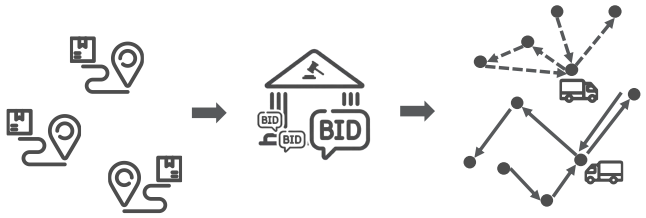

Figure 2: The Auction, Pickup, and Delivery Problem (APDP). Task: Logistic operations optimization. Goal: Maximize profit for a transportation company.
Multiple transportation companies (agents) compete in a market. Each company owns several vehicles that deliver tasks (e.g., parcels) in a given network. Tasks are sold via a reverse first-price sealed-bid auction, i.e., a company's bid corresponds to the amount of money they want to be paid to deliver the task. Higher bids mean more revenue, but bidding too high may result in not getting the auctioned task. A competitive bid depends on (i) the marginal cost of adding the auctioned task to the partial delivery plan, given the already won tasks, (ii) the marginal cost of the opponent, and (iii) other strategic decisions like incurring a loss (bid below your marginal cost) at the beginning in order to reduce the cost of future tasks (better positioning in the market).
After the auction is complete, the company has to determine a plan for its vehicles such that all tasks won by the company are delivered and the total revenue of the company is maximized. The plan is a sequence of pickup and delivery actions, such that vehicle capacity constraints are satisfied. The total revenue of the company is defined as the sum of rewards (won bids paid out by the auction house) minus delivery cost (kilometers driven times cost per kilometer). It is, thus, necessary to bid optimally and compute efficient delivery plans.

competed against human coders and finished second. While there is not much information available about the OpenAI system used in the competition, it was reportedly a custom system (unlike ChatGPT or models available to the public through APIs) [19]. The task was to write code to guide a fleet of robots across a grid, placing barriers to minimize the number of moves [5]. This was an example of a complex optimization problem and required skills beyond code synthesis and knowledge of data structures. The solutions were scored in a competitive way relative to one another. However, there was no direct competition between different opponents within the problem environment.

Our work takes a step further in this direction by increasing the difficulty of evaluation. More specifically, we pick a novel complex domain that : (i) involves complex reasoning in a competitive environment to design an end-to-end solution by combining a number of components (data structures, optimization algorithms, auction mechanisms, modeling the opponents, strategic multi-agent interactions, etc.), (ii) involves generating code for heuristics which can be evaluated based on continuous metrics of quality to compare with other competitors, and (iii) has human coding data available (from pre-LLM era) to compare performance.

3 THE AUCTION, PICKUP, AND DELIVERY PROBLEM (APDP)

The Pickup and Delivery Problem (PDP) is part of a broad and significant class of combinatorial optimization problems central to logistics, transportation, and supply chain management [8, 11] with *large practical value*, as they lie in the *core* of companies like Amazon, DLH, FedEx, etc.[3] Traditionally, these problems set to address the challenge of designing optimal routes for a fleet of vehicles to serve a set of transportation requests (tasks), with the goal of minimizing operational costs while adhering to a complex set of real-world constraints. For more details on the PDP problem and its variants, we refer the interested reader to [8, 11].

Our APDP is a variant of the PDP in which tasks are auctioned in a reverse first-price sealed-bid auction. See Figure 2 for a high level problem description. Transportation companies managed by agents (sellers of services) compete to obtain business from the buyer (task auctioneer) and prices will typically decrease as the companies underbid each other. The APDP has two stages: (i) In the first stage (auction), each task is auctioned one by one. The auction house publishes the details of the task, i.e., the pick-up city, delivery city, and weight. Agents then submit bids, representing the payment that the agent requests for the delivery of the task. Lowest bid wins. Each task is evaluated by agents *locally*, with *no knowledge of future tasks*[4]. Agents need to consider the already won tasks (and potential future tasks) to form bundles. (ii) In the second stage (vehicle routing), each agent has to solve a static PDP problem, efficiently scheduling a fleet of vehicles of *different characteristics* (starting location, capacity, cost per kilometer) to pickup and deliver all won tasks. The overarching goal for each agent is to maximize the profit for the company.

3.1 Problem Formulation

The APDP involves the transportation of tasks (e.g., packages) from an origin location to a corresponding destination in an undirected graph $G(V, E)$ (see Figure 3). Let $T = \{\tau_1, \tau_2, ...\tau_{N_T}\}$ denote the tasks to be auctioned, each with a source and destination in V, and weight w_{τ_i}. The tasks are drawn from a distribution $\Delta(V \times V \times \mathbb{R}_{>0})$, known to the agents. We assume we already know the shortest path between any two cities. Each agent i manages a company C_i operating a fleet of vehicles $V_i = \{v_{i,1}, v_{i,2}, ...v_{i,N_V}\}$. For simplicity, we assume $N_V = 2$ (though the coded solutions can work for $N > 2$). Each vehicle is characterized by its starting location, capacity, and cost per km. I.e., the agents manage a *heterogeneous* set of companies, each operating a *heterogeneous* fleet of vehicles.

At the start of the game, a central auctioneer auctions task sequentially, in a reverse first-price sealed-bid auction. Each participant then has t_{bid} seconds to submit their bid. After each task is auctioned, the agents can observe their opponents' bids. The number of tasks to be auctioned is *not known* to the participants. At the end of the auction, each participant must submit a valid solution, that satisfy certain constraints (described below). Each agent has t_{plan}

[3] PDPs are ubiquitous in real-life outside the described package delivery scenario, in domains such as ride-pooling (dial-a-ride) [17], meal delivery routing [40], supply-chain management for manufacturing companies such as Huawei and Tesla [11].
[4] Agents have access to the task distribution, and they may simulate synthetic future tasks to account for future supply/demand in their planning.

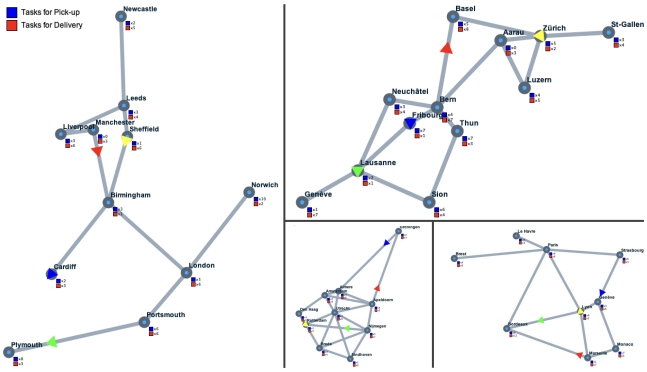

Figure 3: Network Topologies. From top to bottom and left to right we have: Great Britain, Switzerland, the Netherlands, and France. Colored triangles represent vehicles.

seconds to plan. The winner is the agent that maximizes the company's profit (P_i), given by $\sum_{\tau \in \text{won}(i)} b_{i,\tau} - c_{i,\tau}$, where $\sum_{\tau \in \text{won}(i)} b_{i,\tau}$ is the total revenue, calculated by summing the agent's winning bids ($b_{i,\tau}$) for all tasks won by agent i, and $\sum_{\tau \in \text{won}(i)} c_{i,\tau}$ is the total transportation cost (distance driven × cost per kilometer for each vehicle) for delivering all tasks in won(i) according to i's solution.

Core Constraints. A valid solution must satisfy a set of constraints that model operational rules. Specifically: (i) *Capacity:* The cumulative task load at any given time on a vehicle can not exceed its maximum capacity. (ii) *Delivery:* All won tasks must be picked up and delivered. (iii) *Pairing:* A task pickup and its corresponding delivery must be performed by the same vehicle. (iv) *Precedence:* The pickup must precede the delivery for all tasks.

3.2 Challenges in APDP

APDP incorporates challenges from non-cooperative MAS (in stage 1, as they compete against other strategic agents), cooperative MAS (in stage 2, as they collaborative manage a fleet of vehicles), auctions under uncertainty (as they do not know exact valuations or future bundles), and constraint optimization.

The Pickup and Delivery Problem (PDP) is NP-hard [11]. The implication of NP-hardness is that there is no known algorithm that can find a provably optimal solution in a time that grows polynomially with the size of the problem instance (e.g., the number of tasks). This inherent computational complexity creates a fundamental trade-off between solution quality (optimality) and computational time (scalability). Thus, an important characteristic of our proposed benchmark is the ability of an agent to find not the single best solution, but one that *effectively navigates the quality-versus-time trade-off* to meet the demands of practical applications.

In APDP, agents additionally face a *series of interconnected auctions*. An agent's action in one auction, directly affects its state and, thus, strategy space in all subsequent auctions, introducing complex dynamics and strategic thinking.[5]

Some of the major challenges agents face in APDP are as follows:

- Vehicles and their capacity are finite. Committing to tasks changes the profitability of future ones. A rational agent must consider both the *marginal cost* (the additional cost incurred to service a new task), and the *opportunity cost* (the expected value of the best alternative task that is foregone, i.e., potential loss of profit) of its actions.
- The cost of serving a set of tasks is typically *subadditive*; i.e., the cost of serving tasks A and B together may be less than the cost of serving A plus the cost of serving B. This is due to spatial proximity and the fact that we can combine multiple pickups (economies of scope). Thus, an agent might potentially benefit from strategically underbidding to acquire tasks along desirable routes that could potentially substantially decrease the future marginal cost, and thus allowing to recover the lost profit in the future by undercutting the competition.
- Due to the aforementioned synergies between tasks, an agent might realize a significant profit by considering *future tasks* (agents have access to the task distribution) and accounting for expected future combinations (similar to bidding for bundles in combinatorial auctions). This is itself a complex optimization problem as it requires to solve another PDP (NP-hard) for each potential bundle.
- When multiple self-interested agents interact, their behavior is governed not only by their individual cost estimates but also by their *strategic considerations of their opponents' actions*. For example, an agent might have an incentive to bid a value different from their true cost to secure a profit, or to mislead the opponent's beliefs about its competitors (opponents' bids are observable). An agent may also strategically lose an auction to manipulate the state of competitors in future auctions. In general, heterogeneity in bounded rationality among agents gives rise to many possible strategies.
- Estimating the value of a task must respect *real-time* constraints, as agents have limited time to bid and plan.

These challenges imply that one cannot simply employ an off-the-shelf or memorized algorithm as a competitive solution for the APDP, especially when facing such a wide range of potential opponents. These multifaceted challenges – combinatorial optimization, sequential decision-making under uncertainty, and strategic interactions – make APDP a challenging benchmark for the next frontier of code generation.

4 METHODOLOGY

Agents. We evaluated 57 agents in total: 40 LLM-coded agents and 17 human-coded agents.

Tournament. The agents competed in 12 *double all-play-all* tournaments (using 4 network topologies; 3 tournaments with each topology). Agents in a tournament met every other participant in 1v1 matches. In each match, we assigned a company to each agent. Companies have two vehicles with different capacities, cost per kilometer, and starting location. For fairness, we swap the company each agent controls and repeat the match (i.e., each agent plays against the same opponent once controlling company A and once controlling company B). As such, in each tournament we run 3192

[5]There is a gap in theoretical properties of truthful mechanisms in dynamic auctions [32].

matches (57×56, i.e., double all-play-all). In total, we run *38304 matches* (3192 matches $\times 12$ tournaments).

Topologies and Tasks. We run tournaments in 4 different network topologies that correspond to simplified road network abstractions of Switzerland, France, Great Britain, and the Netherlands (see Figure 3). In each match we auctioned *50 tasks*, drawn uniformly at random (a task is defined by its source, destination, and weight).

4.1 Human-coded Agents

We compare the LLM agents against 17 human-coded agents, **developed before the advent of LLMs**. These correspond to 12 agents developed by students of the Intelligent Agents course and 5 baseline agents developed by members of the Artificial Intelligence Laboratory at EPFL.

The Intelligent Agents Course. The Auction, Pickup, and Delivery problem (see Section 3) was given as an assignment in the postgraduate Intelligent Agents course at EPFL. Similarly, members of the Artificial Intelligence Laboratory at EPFL developed the Logist platform we used in this work for our simulations. All code is implemented in Java, and is available as an open-source benchmark at https://panayiotisd.github.io/apdp_bench/.

Baseline Agents. Members of the Artificial Intelligence Laboratory at EPFL developed 5 *simple* baseline agents (prior to the advent of LLMs):

(1) **Naive**: It calculates the distance to the pickup city of the auctioned task from the current city, plus the distance to the destination, and bids distance $\times$ cost per km (with a small added randomness). Then it uses only a single vehicle (disregarding the rest) to sequentially pickup and deliver each won task (i.e., no intelligent batching/reordering, etc.).
(2) **ExpCostFixedBid**: It generates 10 random synthetic tasks (random pickup/delivery cities and weights), and estimates an average marginal cost for these tasks. Then it uses this average as its bid for every auctioned task.
(3) **Honest**: It bids (roughly) the true marginal cost of adding the auctioned task to its current fleet schedule. The marginal cost calculation is done by looking across all vehicles and returning the minimum marginal cost of inserting this task given the current assignments.
(4) **ModelOpponent**: It calculates the marginal cost (calculated like the Honest) for both itself and its opponent, and bids the maximum between the two marginal costs. For the opponents marginal cost calculation, it keeps track of the opponent's won tasks and simulates a shadow fleet using its own fleet specs as a proxy.
(5) **RiskSeeking**: It uses an exponential schedule that shifts over time from a rough prior (calculated using synthetic tasks like ExpCostFixedBid) to the current marginal cost (calculated like the Honest). Additionally, it keeps track of the opponent's won tasks and using a shadow fleet (like ModelOpponent), and calculates the same blended cost (using the aforementioned exponential schedule). Finally, it bids the maximum between its own and the opponent's blended cost.

Student Agents. We have chosen 12 student agents developed (in 2-3 weeks time, along with their normal coursework) as part of the Intelligent Agents course in *2020* (2 years prior to the introduction of ChatGPT). These agents were chosen as follows. At the end of the Intelligent Agents course, students competed in a *single-elimination tournament*. We chose the top 8 agents from the 2020 tournament. Note that due to the nature of single-elimination tournaments, these top 8 agents are not necessarily the top human-coded agents overall (in fact, some of them fail to beat the Baseline Agents). We also selected 4 additional student agents that performed well (highest number of wins) against the aforedescribed Baseline Agents.

4.2 LLM-coded Agents

We employed 4 different state-of-the-art LLMs: OpenAI's **GPT-5 Thinking**, Google's **Gemini 2.5 Pro**, Anthropic's **Claude Opus 4.1**, and DeepSeek's **DeepThink R1**. We also used 5 different prompting strategies, as explained below. Each LLM was prompted twice with each of the 5 prompting strategies, for a total of *40 LLM-coded agents*.

Prompting Strategies. We used the following strategies:

(1) **Author Prompt #1 (A1):** Prompt generated by one of the authors, as detailed in the following paragraph.
(2) **Author Prompt #2 (A2):** Prompt generated by a different author.
(3) **Iterative Refinement (IR):** We evaluate the agent generated using prompt A1 in *self-play*, and we provide the scores as input back to the respective LLM (the one that generated the A1 agent), along with the following addition to the prompt: 'Would you like to improve anything, or shall we enter the tournament? We need to win!'.[6]
(4) **LLM as a Critic (CR):** We feed the code generated by an LLM (using the A1 prompt) to GPT-5 Thinking (the top performing LLM in our test), along with a short problem description, and ask GPT-5 to 'provide a list of improvements to make sure that the agent maximizes profit and wins the competition!'. Then we provide the suggested improvements to the original LLM (the one that generated the A1 agent) and ask to implement them.
(5) **GPT-5 generated (optimized) prompt (GEN):** We gave the A1 prompt to GPT-5 Thinking (the top performing LLM in our test) and asked it to optimize the prompt. Then, we used the resulting prompt for code generation.

The supplement contains the complete prompts. Below we provide a description of one of the prompts as an example.

Description of Author Prompt #1. The prompt was written to contain the same information as the students received in the Intelligent Agents course project. This is because we want to investigate if LLMs are capable of competing with humans (students), when they are provided with same instructions. This would also

[6]Note that in our application the feedback signal is weak, compared to iterative refinement in other coding benchmarks. An interesting avenue to investigate in future work is whether asking the LLM to include prints of debug information (as a human software developer would do) would make a difference.

correspond to a modern-day student using an LLM to complete their project, or people using vibe-coding approaches.

The A1 prompt aggregated information from the project description, the project presentation slides, and relevant pitfalls we discussed with students during the lab sessions over the years the authors were part of the course. The information was cleaned and structured according to known best practices[7]. In short, the A1 prompt contained the following information:

The LLM is assigned the role of an expert AI/ML coder. The prompt then provides a high level problem description/summary (similar to what is shown in Figure 2). Then the prompt describes in detail the vehicle planning problem formulated as a constraint satisfaction problem, using LaTeX to describe variables, constraints, and the cost function. The prompt stresses that the LLM does not need to follow this formulation; instead it may follow any alternative approach. It further stresses that the LLM must find the plan that maximizes the revenue of the company. This is followed by the description of the auction rules, and the strategic decisions the LLM should consider. The prompt continues with a list of tasks the LLM needs to complete, the competition rules, and the *template code* that the LLM needs to complete. Finally, at the very end, the prompt summarizes key points and tasks, as follows:

- You need to implement an agent that (i) competes against other agents in a tournament, (ii) optimally bids for tasks, and (iii) optimally delivers the won tasks.
- Tasks are allocated via a first-price reverse auction. In other words, you are bidding on how much money the auction house will pay you to deliver a task.
- You have to deliver all the tasks you win.
- The total revenue is defined as the sum of rewards (won bids paid by the auction house) minus delivery cost (km driven times cost per km). As such, bidding optimally and computing an efficient plan are of paramount importance.
- You do not have to implement Stochastic Local Search for the delivery planning. Implement the best option you can. Make sure that your agent obeys the time limits.
- Most importantly, you need to implement the best possible solution, the one that maximizes profit and wins the competition!

Debugging. We tested all LLMs on both self-play and tournament conditions until all bugs we could identify got resolved. Each LLM was responsible for fixing the bugs in its code (by prompting the LLM with the error info). We observed a *minimal number of syntactic errors*, but a *significant number of semantic bugs*. Notable examples include: (i) Not respecting the time-out limits (each agent has a fixed time for planning and bidding) despite being given the template code that implements this functionality and being explicitly instructed to adhere to the time limits. (ii) Agents not picking up and/or not delivering tasks they have won in the auction, despite being explicitly instructed to do so. We observed that this was due to either ignoring instructions, or due to logic errors (e.g., removing tasks from a full vehicle during re-planning and then omitting to plan for those). (iii) Violating capacity constraints, where the agents would try to pickup a task that the vehicle could not carry.

[7] Author generated prompts (strategies #1 and #2) followed best practices, e.g., [4, 37].

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[8].

5 RESULTS

5.1 Preliminary Results

Before diving into the results of the tournament, we discuss some insights from a preliminary experiment in which we tested the LLMs' ability to solve simpler variants of the APDP problem. In these variants, the agents do not participate in an auction, and instead solve simpler versions of only the task planning part. These variants are as follows. (i) A simple *Reactive* agent which moves from city to city and if there is a task available to pick up at the current city (tasks drawn from a known distribution) it has to decide whether to accept it and immediately deliver it, or reject it in which case it will be lost and the agent will move on to another city. This variant involves modeling the problem as an MDP and solving it using value iteration. Then the agent selects actions based on a learned state-action table. (ii) A *Deliberative* agent which, unlike the reactive one, it knows in advance the list of tasks that must be delivered. As such, the goal is to construct a plan that guarantees the optimal delivery. It does so by modeling all possible states and using BFS and A* to find the optimal plan. Finally, (iii) a *Centralized* agent, which faces the PDP constraint optimization problem (Section 3), but there is no auction or multiple companies involved. The problem involves implementing the Stochastic Local Search algorithm.

Reactive. All LLMs solved this test-case correctly on the first try, with minimal syntax errors (that they resolved when provided with the compiler error logs).

Deliberative. In this test-case, we observed a surprising result: 3 out of the 4 LLMs (all but GPT-5) **did not implement an admissible heuristic for A*** (the first time), despite being explicitly asked to do so in the prompt (emphasized in multiple points and in the summary of the task at the end of the prompt). Without an admissible heuristic A* overestimates the cost of reaching the goal, and thus prunes optimal paths. It is a fundamental requirement for A* to work, and the first thing any student learns about A*.

In general, we observed very inconsistent behavior. For example, one of the Claude agents started with an admissible heuristic, but when asked to fix the bugs because the solution was not optimal, it changed the heuristic to an inadmissible one[9]. In another example, we used GPT4o as a critic, but Claude did not manage to fix the issues GPT4o raised, and the heuristic was still inadmissible[10].

[8] A related observation was made in [13], where authors find that 'repeated sampling from the model is a surprisingly effective strategy for producing working solutions to difficult prompts'.

[9] We did not specify what is the bug, but instead let the LLM figure it out itself.

[10] Contrary to that, GPT4o was able to update Claude's code and fix the issues.

Table 1: Table of results across the 12 double all-play-all tournaments (4 network topologies, 3 tournaments with each topology, totaling almost 40k matches). Each agent in a tournament plays 112 matches, thus the upper limit for the Avg #Wins / Tour and Avg #Losses / Tour is 112. SD = Standard deviation. Human-coded agents in bold. The naming scheme for the LLMs is as follows. The first letter in the parenthesis refers to the model: O = OpenAI's GPT-5 Thinking, G = Google's Gemini 2.5 Pro, A = Anthropic's Claude Opus 4.1, and D = DeepSeek's DeepThink R1. The next two letters refer to the prompting scheme (see Section 4). The last number refers to the 1st or 2nd agent we generated (we run each LLM/prompt combination twice, see Section 4).

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

Even when LLMs were providing admissible heuristics, it is worth considering that not all heuristics are equally good. Some allow for more aggressive pruning, resulting in a significantly faster solution. For example, DeepThink R1 was the only LLM that opted to implement a Minimum Spanning Tree based heuristic, which is significantly tighter that any of the heuristics implemented by the rest of the LLMs, allowing it to gracefully scale to larger problem sizes. As a result, DeepThink R1's agent was *one order of magnitude* faster than both GPT's and Claude's.

The A* algorithm is one of the most established and well known classic algorithms in computer science and artificial intelligence. Thus, it is surprising that state-of-the-art LLMs struggled to implement an admissible heuristic (not to mention their inability to implement a tight heuristic).

Centralized. Similar to the Deliberative test-case, the LLMs were able to produce syntax-bug-free code, but we often observed suboptimal design decisions. For example, in this test-case the LLMs have to schedule multiple vehicles. We observed situations where the resulting agent would utilize only one vehicle from its fleet, despite being specifically prompted to find the optimal solution, and despite the LLM itself noticing this inefficiency and suggesting to improve it: 'Let me know if you want to further improve the solution by [...] supporting multiple pickups before delivery'. Finally, we compared the LLM-coded agents to a simple template code,[11] and observed a large variation in profit, ranging from 19% higher to 194% lower (compared to the template code).

5.2 Tournament Results

The observations from the preliminary evaluation indicate that 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. We now demonstrate the implications of these weaknesses on the full APDP problem variant.

Table 1 contains the scores for the 12 double all-play-all tournaments (~*40k matches*). We report the average number of wins and losses per tournament (bounded by 112 – the number of matches for each agent per tournament), the standard deviation, the total number of wins and losses across all tournaments, and the win rate.

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, see Section 4). 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*.

[11] A very basic implementation of SLS, without any optimizations, instead designed to be easily read and understood by the students of the Intelligent Agents course. It starts by assigning all tasks to the largest vehicle, and then generates neighbors by randomly changing the task order inside a vehicle or moving a task to a different vehicle.

For reference, Student 9 crashed on average 15,33 ± 13,44 times per tournament (184 in total across tournaments), Student 11 6,17 ± 0,94 times (74), Student 4 5,75 ± 1,06 times (69), Student 7 3,42 ± 1,08 times (41), Student 8 2,00 ± 0,00 times (24), Student 10 2,00 ± 0,00 times (24), Student 5 0,42 ± 0,90 times (5), Student 1 0,25 ± 0,62 times (3), Student 6 0,08 ± 0,29 times (1).

5.3 Improving the Tournament Winner?

It is evident from all the results so far that LLMs underperform compared to student-coded agents in our APDP benchmark. In a further experiment, we asked GPT-5 Pro (the best performing LLM in our benchmark) to improve the code of the winning agent (Student 1). We provided a prompt with the description of the problem (highlighting that the agent has to bid optimally, plan efficient vehicle routes, maximize profit, obey all time constraints, and win the tournament), the code of Student 1, and asked it to implement any improvements it deems necessary to win the tournament. Surprisingly, the resulting agent **ranked 10th** (just below Student 7), with an average number of wins of 89.667. In other words, **the LLM's 'optimizations' resulted in losing 9 spots in the leaderboard**.

Many benchmarks are often accompanied with *data contamination concerns* (i.e., the correct answers were already in the training data). On the other hand, this experiment shows that, 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.

6 IMPLICATIONS, LIMITATIONS, AND FUTURE WORK

We considered the setting of using LLMs as a tool to generate code (vibe coding), with the aim to evaluate the performance of a representative sample of various LLMs (both paid and free) from various companies, ones that most people would have access to. This is the *first* comparison of LLMs vs. humans in this new frontier in automated code generation.

Our work is *not* concerned with finding the optimal prompting strategy, the best LLM, or to create a leader-board of LLMs. Our conclusions are bounded by a single, albeit rich, domain (logistics PDP). We also do not claim that LLM performance observed in this paper is the optimal performance the best LLM can ever achieve in this setting with any prompt. It is clear that some LLMs are better at some tasks than others, and there are even dedicated models, specifically fine-tuned for coding. Just like professional software engineers could presumably be better than the students in this study, and some engineers with more experience would be better than others. But the *wide range* of employed LLMs and prompting strategies allows us to evaluate, for the first time, *vibe coding* as a means of solving complex optimization problems. The appeal of vibe coding is that it empowers people of all technical backgrounds, not just those who are up to date with the latest prompting recommendations or those that have access to custom models. This is also the reason for not directly reporting the name of each LLM in Table 1, but instead including it as secondary information inside a parenthesis, as our aim was to investigate what the state-of-the-art

LLMs can do overall compared to humans, instead of creating a leaderboard of LLMs.

Our results highlight important limitations of LLM code generation, most notably their limited reasoning and planning capabilities while generating code (see also [29]). 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. The results call for the development of a suite of next generation code synthesis benchmarks, grounded in real-world settings, involving aspects of both multi-agent competition and collaboration.

7 CONCLUSION

In this work, we investigate whether LLMs' strong reported performance on unit-tests translates to competence in solving real-world software engineering problems, ones requiring planning, optimization, and advanced algorithm design. The undeniably impressive performance of LLMs often gives the impression that vibe coding can spin up software on demand without requiring any technical background by the user. Yet, results from our Auction, Pickup, and Delivery Problem (APDP) benchmark (a multi-agent, strategic-reasoning-driven optimization benchmark), demonstrate a clear superiority of human-coded agents in problems requiring long-horizon planning, opponent reasoning, and strategic optimization. Human-coded agents (developed by graduate students before the advent of LLMs) consistently occupy the top five positions, and the majority of the LLM-coded agents (33/40) lose to simple baselines. These results are a call to explore a new frontier in code synthesis; shifting the goal from *code that compiles* to *code that competes*.

ACKNOWLEDGMENTS

We are grateful to Professor Emeritus Boi Faltings and the members of the Artificial Intelligence Laboratory at EPFL who contributed in the development of the Intelligent Agents course over the years. We also thank the students who contributed their code as baselines. The list of contributors is available at: https://panayiotisd.github.io/apdp_bench/.

REFERENCES

[1] Josh Achiam, Steven Adler, Sandhini Agarwal, Lama Ahmad, Ilge Akkaya, Florencia Leoni Aleman, Diogo Almeida, Janko Altenschmidt, Sam Altman, Shyamal Anadkat, et al. 2023. Gpt-4 technical report. *arXiv preprint arXiv:2303.08774* (2023).

[2] Dario Amodei. 2024. Dario Amodei: Anthropic CEO on Claude, AGI & the Future of AI & Humanity | Lex Fridman Podcast. https://youtu.be/ugvHCXCOmm4?t=8987. Accessed: 2025-09-12.

[3] Anthropic. [n.d.]. The Claude 3 Model Family: Opus, Sonnet, Haiku. https://www-cdn.anthropic.com/de8ba9b01c9ab7cbabf5c33b80b7bbc618857627/Model_Card_Claude_3.pdf. Accessed: 2025-09-12.

[4] Anthropic. 2025. Prompt engineering overview. https://docs.claude.com/en/docs/build-with-claude/prompt-engineering/overview. Accessed: 2025-10-01.

[5] AtCoder. 2025. Group Commands and Wall Planning. https://atcoder.jp/contests/awtf2025heuristic/tasks/awtf2025heuristic_a. Accessed: 2025-09-22.

[6] Jacob Austin, Augustus Odena, Maxwell Nye, Maarten Bosma, Henryk Michalewski, David Dohan, Ellen Jiang, Carrie Cai, Michael Terry, Quoc Le, et al. 2021. Program synthesis with large language models. *arXiv preprint arXiv:2108.07732* (2021).

[7] Joel Becker, Nate Rush, Elizabeth Barnes, and David Rein. 2025. Measuring the impact of early-2025 AI on experienced open-source developer productivity. *arXiv preprint arXiv:2507.09089* (2025).

[8] Gerardo Berbeglia, Jean-François Cordeau, Irina Gribkovskaia, and Gilbert Laporte. 2007. Static pickup and delivery problems: a classification scheme and survey. *Top* 15, 1 (2007), 1–31.

[9] Bloomberg. 2025. AI Coding Assistant Cursor Draws a Million Users Without Even Trying. https://www.bloomberg.com/news/articles/2025-04-07/cursor-an-ai-coding-assistant-draws-a-million-users-without-even-trying. Accessed: 2025-09-23.

[10] Noam Brown. 2025. Twitter(X) Thread. https://x.com/polynoamial/status/1918746853866127700. Accessed: 2025-09-12.

[11] Junchuang Cai, Qingling Zhu, Qiuzhen Lin, Lijia Ma, Jianqiang Li, and Zhong Ming. 2023. A survey of dynamic pickup and delivery problems. *Neurocomputing* 554 (2023), 126631. https://doi.org/10.1016/j.neucom.2023.126631

[12] Satish Chandra and Maxim Tabachnyk. 2024. AI in software engineering at Google: Progress and the path ahead. https://research.google/blog/ai-in-software-engineering-at-google-progress-and-the-path-ahead/. Accessed: 2025-09-23.

[13] Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Ponde De Oliveira Pinto, Jared Kaplan, Harri Edwards, Yuri Burda, Nicholas Joseph, Greg Brockman, et al. 2021. Evaluating large language models trained on code. *arXiv preprint arXiv:2107.03374* (2021).

[14] Wei-Lin Chiang, Lianmin Zheng, Ying Sheng, Anastasios Nikolas Angelopoulos, Tianle Li, Dacheng Li, Banghua Zhu, Hao Zhang, Michael Jordan, Joseph E Gonzalez, et al. 2024. Chatbot arena: An open platform for evaluating llms by human preference. In *Forty-first International Conference on Machine Learning*.

[15] Neil Chowdhury, James Aung, Chan Jun Shern, Oliver Jaffe, Dane Sherburn, Giulio Starace, Evan Mays, Rachel Dias, Marwan Aljubeh, Mia Glaese, Carlos E. Jimenez, John Yang, Leyton Ho, Tejal Patwardhan, Kevin Liu, and Aleksander Madry. 2025. Introducing SWE-bench Verified. https://openai.com/index/introducing-swe-bench-verified/. Accessed: 2025-09-12.

[16] Zheyuan Kevin Cui, Mert Demirer, Sonia Jaffe, Leon Musolff, Sida Peng, and Tobias Salz. 2025. The Effects of Generative AI on High-Skilled Work: Evidence from Three Field Experiments with Software Developers. *Available at SSRN 4945566* (2025).

[17] Panayiotis Danassis, Marija Sakota, Aris Filos-Ratsikas, and Boi Faltings. 2022. Putting ridesharing to the test: efficient and scalable solutions and the power of dynamic vehicle relocation. *Artificial Intelligence Review* 55, 7 (01 Oct 2022), 5781–5844. https://doi.org/10.1007/s10462-022-10145-0

[18] Xiang Deng, Jeff Da, Edwin Pan, Yannis Yiming He, Charles Ide, Kanak Garg, Niklas Lauffer, Andrew Park, Nitin Pasari, Chetan Rane, et al. 2025. SWE-Bench Pro: Can AI Agents Solve Long-Horizon Software Engineering Tasks? *arXiv preprint arXiv:2509.16941* (2025).

[19] Hannah Devlin. 2025. Competition shows humans are still better than AI at coding. https://www.theguardian.com/technology/2025/jul/26/competition-shows-humans-are-still-better-than-ai-at-coding-just. Accessed: 2025-09-22.

[20] Yihong Dong, Xue Jiang, Jiaru Qian, Tian Wang, Kechi Zhang, Zhi Jin, and Ge Li. 2025. A Survey on Code Generation with LLM-based Agents. *arXiv preprint arXiv:2508.00083* (2025).

[21] Mingzhe Du, Anh Tuan Luu, Bin Ji, Qian Liu, and See-Kiong Ng. 2024. Mercury: A code efficiency benchmark for code large language models. *Advances in Neural Information Processing Systems* 37 (2024), 16601–16622.

[22] P. Gauthier. [n.d.]. Aider Polyglot Coding Leaderboard. https://aider.chat/docs/leaderboards/. Accessed: 2025-09-12.

[23] Google. 2025. Gemini 2.5: Pushing the Frontier with Advanced Reasoning, Multimodality, Long Context, and Next Generation Agentic Capabilities. https://storage.googleapis.com/deepmind-media/gemini/gemini_v2_5_report.pdf. Accessed: 2025-09-12.

[24] Antoine Grosnit, Alexandre Maraval, Refinath S N, Zichao Zhao, James Doran, Giuseppe Paolo, Albert Thomas, Jonas Gonzalez, Abhineet Kumar, Khyati Khandelwal, Abdelhakim Benechehab, Hamza Cherkaoui, Youssef Attia El-Hili, Kun Shao, Jianye Hao, Jun Yao, Balázs Kégl, Haitham Bou-Ammar, and Jun Wang. 2025. Kolb-Based Experiential Learning for Generalist Agents with Human-Level Kaggle Data Science Performance. arXiv:2411.03562 [cs.LG] https://arxiv.org/abs/2411.03562

[25] Dan Hendrycks, Steven Basart, Saurav Kadavath, Mantas Mazeika, Akul Arora, Ethan Guo, Collin Burns, Samir Puranik, Horace He, Dawn Song, and Jacob Steinhardt. 2021. Measuring Coding Challenge Competence With APPS. In *Proceedings of the Neural Information Processing Systems Track on Datasets and Benchmarks*, J. Vanschoren and S. Yeung (Eds.), Vol. 1. https://datasets-benchmarks-proceedings.neurips.cc/paper_files/paper/2021/file/c24cd76e1ce41366a4bbe8a49b02a028-Paper-round2.pdf

[26] Nam Huynh and Beiyu Lin. 2025. Large Language Models for Code Generation: A Comprehensive Survey of Challenges, Techniques, Evaluation, and Applications. *arXiv preprint arXiv:2503.01245* (2025).

[27] Naman Jain, King Han, Alex Gu, Wen-Ding Li, Fanjia Yan, Tianjun Zhang, Sida Wang, Armando Solar-Lezama, Koushik Sen, and Ion Stoica. 2025. LiveCodeBench: Holistic and Contamination Free Evaluation of Large Language Models for Code. In *The Thirteenth International Conference on Learning Representations*. https://openreview.net/forum?id=chfJJYC3iL

[28] Carlos E Jimenez, John Yang, Alexander Wettig, Shunyu Yao, Kexin Pei, Ofir Press, and Karthik R Narasimhan. 2024. SWE-bench: Can Language Models

Resolve Real-world Github Issues?. In *The Twelfth International Conference on Learning Representations*. https://openreview.net/forum?id=VTF8yNQM66

[29] Subbarao Kambhampati. 2024. Can large language models reason and plan? *Annals of the New York Academy of Sciences* 1534, 1 (2024), 15–18.

[30] Yujia Li, David Choi, Junyoung Chung, Nate Kushman, Julian Schrittwieser, Rémi Leblond, Tom Eccles, James Keeling, Felix Gimeno, Agustin Dal Lago, et al. 2022. Competition-level code generation with alphacode. *Science* 378, 6624 (2022), 1092–1097.

[31] Fang Liu, Yang Liu, Lin Shi, Houkun Huang, Ruifeng Wang, Zhen Yang, Li Zhang, Zhongqi Li, and Yuchi Ma. 2024. Exploring and evaluating hallucinations in llm-powered code generation. *arXiv preprint arXiv:2404.00971* (2024).

[32] Johan Los, Frederik Schulte, Matthijs TJ Spaan, and Rudy R Negenborn. 2022. Strategic Bidding in Decentralized Collaborative Vehicle Routing. In *International Conference on Dynamics in Logistics*. Springer, 261–274.

[33] Iain Martin. 2025. Vibe Coding Turned This Swedish AI Unicorn Into The Fastest Growing Software Startup Ever. https://www.forbes.com/sites/iainmartin/2025/07/23/vibe-coding-turned-this-swedish-ai-unicorn-into-the-fastest-growing-software-startup-ever/. Accessed: 2025-09-14.

[34] Samuel Miserendino, Michele Wang, Tejal Patwardhan, and Johannes Heidecke. 2025. SWE-Lancer: Can Frontier LLMs Earn $1 Million from Real-World Freelance Software Engineering? *arXiv preprint arXiv:2502.12115* (2025).

[35] Hussein Mozannar, Valerie Chen, Mohammed Alsobay, Subhro Das, Sebastian Zhao, Dennis Wei, Manish Nagireddy, Prasanna Sattigeri, Ameet Talwalkar, and David Sontag. 2025. The RealHumanEval: Evaluating Large Language Models' Abilities to Support Programmers. *Transactions on Machine Learning Research* (2025). https://openreview.net/forum?id=hGaWq5Buj7

[36] Alexander Novikov, Ngân Vu, Marvin Eisenberger, Emilien Dupont, Po-Sen Huang, Adam Zsolt Wagner, Sergey Shirobokov, Borislav Kozlovskii, Francisco JR Ruiz, Abbas Mehrabian, et al. 2025. Alphaevolve: A gemini-powered coding agent for designing advanced algorithms. https://deepmind.google/discover/blog/alphaevolve-a-gemini-powered-coding-agent-for-designing-advanced-algorithms/.

[37] OpenAI. 2025. Prompting guidance. https://platform.openai.com/docs/guides/latest-model#prompting-guidance. Accessed: 2025-10-01.

[38] Ruizhong Qiu, Weiliang Will Zeng, James Ezick, Christopher Lott, and Hanghang Tong. 2025. How efficient is LLM-generated code? A rigorous & high-standard benchmark. In *The Thirteenth International Conference on Learning Representations*. https://openreview.net/forum?id=suz4utPr9Y

[39] Shuo Ren, Daya Guo, Shuai Lu, Long Zhou, Shujie Liu, Duyu Tang, Neel Sundaresan, Ming Zhou, Ambrosio Blanco, and Shuai Ma. 2020. Codebleu: a method for automatic evaluation of code synthesis. *arXiv preprint arXiv:2009.10297* (2020).

[40] Damian Reyes, Alan Erera, Martin Savelsbergh, Sagar Sahasrabudhe, and Ryan O'Neil. 2018. The meal delivery routing problem. *Optimization Online* 6571, 2018 (2018), 2018.

[41] Manley Roberts, Himanshu Thakur, Christine Herlihy, Colin White, and Samuel Dooley. 2024. To the Cutoff... and Beyond? A Longitudinal Perspective on LLM Data Contamination. In *The Twelfth International Conference on Learning Representations*. https://openreview.net/forum?id=m2NVG4Htxs

[42] Runlopp. 2025. Assessing AI Code Quality: 10 Critical Dimensions for Evaluation. https://www.runloop.ai/blog/assessing-ai-code-quality-10-critical-dimensions-for-evaluation. Accessed: 2025-09-12.

[43] Ryan J. Salva. 2025. How are developers using AI? Inside our 2025 DORA report. https://blog.google/technology/developers/dora-report-2025/. Accessed: 2025-09-25.

[44] LMArena Team. [n.d.]. Webdev Arena. https://web.lmarena.ai/leaderboard. Accessed: 2025-09-12.

[45] Colin White, Samuel Dooley, Manley Roberts, Arka Pal, Benjamin Feuer, Siddhartha Jain, Ravid Shwartz-Ziv, Neel Jain, Khalid Saifullah, Sreemanti Dey, Shubh-Agrawal, Sandeep Singh Sandha, Siddartha Venkat Naidu, Chinmay Hegde, Yann LeCun, Tom Goldstein, Willie Neiswanger, and Micah Goldblum. 2025. LiveBench: A Challenging, Contamination-Limited LLM Benchmark. In *The Thirteenth International Conference on Learning Representations*. https://openreview.net/forum?id=sKYHBTAxVa

[46] Sangyeop Yeo, Yu-Seung Ma, Sang Cheol Kim, Hyungkook Jun, and Taeho Kim. 2024. Framework for evaluating code generation ability of large language models. *Etri Journal* 46, 1 (2024), 106–117.

[47] Doron Yeverechyahu, Raveesh Mayya, and Gal Oestreicher-Singer. 2024. The impact of large language models on open-source innovation: Evidence from github copilot. *arXiv preprint arXiv:2409.08379* (2024).

[48] Linghao Zhang, Shilin He, Chaoyun Zhang, Yu Kang, Bowen Li, Chengxing Xie, Junhao Wang, Maoquan Wang, Yufan Huang, Shengyu Fu, et al. 2025. SWE-bench Goes Live! *arXiv preprint arXiv:2505.23419* (2025).

[49] Li Zhong and Zilong Wang. 2024. Can llm replace stack overflow? a study on robustness and reliability of large language model code generation. In *Proceedings of the AAAI conference on artificial intelligence*, Vol. 38. 21841–21849.

[50] Terry Yue Zhuo, Vu Minh Chien, Jenny Chim, Han Hu, Wenhao Yu, Ratnadira Widyasari, Imam Nur Bani Yusuf, Haolan Zhan, Junda He, Indraneil Paul, Simon Brunner, Chen GONG, James Hoang, Armel Randy Zebaze, Xiaoheng Hong, Wen-Ding Li, Jean Kaddour, Ming Xu, Zhihan Zhang, Prateek Yadav, Naman Jain, Alex Gu, Zhoujun Cheng, Jiawei Liu, Qian Liu, Zijian Wang, David Lo, Binyuan Hui, Niklas Muennighoff, Daniel Fried, Xiaoning Du, Harm de Vries, and Leandro Von Werra. 2025. BigCodeBench: Benchmarking Code Generation with Diverse Function Calls and Complex Instructions. In *The Thirteenth International Conference on Learning Representations*. https://openreview.net/forum?id=YrycTjllL0