How Artificial General Intelligence Might Train Itself

David A. Noever
PeopleTec, Inc.
Huntsville, AL
david.noever@peopletec.com

Forrest McKee
PeopleTec, Inc.
Huntsville, AL
forrest.mckee@peopletec.com

J. Wesley Regian
PeopleTec, Inc.
Huntsville, AL
wes.regian@peopletec.com

ABSTRACT

Ambitious projections suggest that artificial general intelligence (AGI) might emerge within the next decade, raising questions about how to train such broad intelligence beyond simply scaling up model parameters. This paper explores the hypothesis that one key bottleneck for AGI is not model size but the availability of unique, high-quality, and potentially infinite or streaming data. We survey a series of simulation-based learning case studies – including autonomous code bug-fixing, self-optimizing route planning, and adversarial scene anomaly detection – to illustrate how AI agents can generate domain-specific knowledge through self-play and machine-versus-machine training. These cases draw inspiration from successful self-learning paradigms like DeepMind's AlphaZero and related self-supervised breakthroughs, where AI systems achieved superhuman performance with zero human-generated training content. We discuss the challenges of scaling such simulations, including scorecard-driven verification of learning, ensuring content diversity, and cultivating synthetic domain expertise that transfers to real-world tasks. We frame these machine-generated simulations as a new class of training artifacts that can augment human–AI collaborative learning. Finally, we propose a conceptual framework – a 'periodic table' of AGI training strategies – that categorizes methods such as human annotation, expert bootstrapping, self-play, curriculum learning, co-evolution, and autonomous agents. The periodic table highlights less tried methods and helps identify gaps in this hybrid approach. The findings suggest that yes – AI can, to a significant extent, train itself by creating an endless curriculum of challenges, although strategic integration of human knowledge and careful simulation design remain highly curated today.

ABOUT THE AUTHORS

David Noever has 27 years of research experience with NASA and Department of Defense in machine learning and data mining. He received his Ph.D. from Oxford University, as a Rhodes Scholar, in theoretical physics and B.Sc. from Princeton University. While at NASA, he was named 1998 Discover Magazine's "Inventor of the Year," for the novel development of computational biology software and internet search robots. He has authored more than 100 peer-reviewed scientific research articles and book chapters. His primary research centers on machine learning, algorithms, and data mining for analytics, intelligence, and novel metric generation.

Forrest McKee has AI research experience with the Department of Defense in cybersecurity, large language models and reinforcement learning. He received his Bachelor's (BS) and Master's (MSE) from the University of Alabama, Huntsville, in Systems Engineering. His research interests center on technical differences between machine and human perception in language, vision, and audio/video.

J. Wesley Regian has 32 years of experience in cognitive performance modeling and knowledge-based software technology development, primarily for military application with AFRL, AFOSR, and DARPA. His work has supported over 50 fielded systems. He has published over 100 papers on intelligence analysis, human terrain modeling, knowledge representation, and human learning and memory. Dr. Regian was a National Research Council research adviser for 10 years and Senior Scientist for Knowledge Based Systems at the US Air Force Armstrong Research Laboratory.

How Artificial General Intelligence Might Train Itself

David A. Noever
PeopleTec, Inc.
Huntsville, AL
david.noever@peopletec.com

Forrest McKee
PeopleTec, Inc.
Huntsville, AL
forrest.mckee@peopletec.com

J. Wesley Regian
PeopleTec, Inc.
Huntsville, AL
wes.regian@peopletec.com

INTRODUCTION

The development of artificial general intelligence (AGI) is widely believed to require training at a scale and breadth far beyond current AI systems (Kurshan, 2023). State-of-the-art large language models (LLMs) and vision models, for example, have already consumed massive datasets consisting of most of the human knowledge available on the internet. Recent estimates suggest that on the order of 15 trillion tokens of high-quality text have been used to train frontier models, and the supply of new human-generated text is quickly diminishing (Tibau, 2024). Interestingly, this quantity of training inputs is roughly 100,000 times what a human being needs to learn a language. Even with aggressive data collection, public web data might only provide at most a few times more (perhaps 25–30 trillion tokens) before it is exhausted (Tibau, 2024). In short, we may soon "run out" of readily usable human data for training ever-larger models. As one analysis concludes, *"with significant effort, we might expand to 25 or 30 trillion [tokens], but not much more… Historically, each model generation required 10x more data. Therefore, new ideas to 'feed' AI learning will be needed soon*" (Tibau, 2024). This looming data bottleneck has shifted attention from model-centric scaling (simply increasing parameters; Achiam, et al. 2023) to data-centric strategies for training the next generation of AI systems.

One promising approach is suggested by a simple question: *Can AI train itself?* Rather than relying exclusively on human-curated data, can an AI system generate its own endless stream of training experiences to become more and more knowledgeable and capable? In other words, if human knowledge captured in text, images, and demonstrations is finite, perhaps machines can create synthetic knowledge sources that are effectively infinite. This idea draws inspiration from how humans learn through practice and simulation. For example, pilots train in flight simulators that can generate countless flight scenarios; soldiers engage in war-game simulations; and children learn through self-directed play. Likewise, an AI might continually practice in simulated environments of its own making, constantly producing new challenges and learning opportunities beyond what static human data provides.

There is growing evidence that AI systems can indeed improve by training themselves in interactive environments. In games and other constrained domains, self-play has emerged as a powerful paradigm for self-training. The landmark achievement was DeepMind's AlphaGo Zero, which mastered the game of Go *tabula rasa* – from scratch, without any human game records – by playing millions of games against itself and using reinforcement learning (RL) to improve with each iteration (Silver et al., 2017). Within days, AlphaGo Zero surpassed the version of AlphaGo that had defeated a world champion, despite having learned entirely from self-play with zero human-provided examples (Silver et al., 2017). This was soon generalized into the AlphaZero system, which achieved superhuman performance in multiple board games (chess, Go, and shogi) using the same self-play RL algorithm, again starting from random play and given no domain knowledge beyond the game rules (Silver et al., 2017). Remarkably, AlphaZero reached elite play in these games in under 24 hours of training, purely by playing against itself (Silver et al., 2017).

The success of AlphaZero hinted that data might be a bigger key to intelligence than model size: it used a relatively modest neural network (with millions, not billions, of parameters) but generated a massive corpus of training data through self-play. Each game played was new data that helped it refine its strategy. Following this, other examples quickly emerged. DeepMind's AlphaStar agent achieved Grandmaster level in the real-time strategy game StarCraft II via multi-agent self-play training, ultimately ranking above 99.8% of human players (Vinyals et al., 2019). OpenAI's Five system similarly trained entirely through self-play to beat the world champions at the complex video game Dota 2, practicing the equivalent of 180 years of games against itself every day (Berner, et al., 2019). In these cases, once the environment (the game simulator) and the objective (win/loss score) were defined, the AI agents generated an effectively infinite training set by playing millions of rounds against copies of themselves. No new human data was needed – the AI became both the student and the teacher, iteratively improving its performance.

These achievements demonstrate that machine-generated training data via simulation and self-play can rival or surpass the quality of human-curated data in certain domains. The paradigm has rapidly expanded beyond games. For instance, self-play and simulation-based learning have been applied to robotics and navigation tasks, where virtual environments allow robots to teach themselves skills that would be costly or dangerous to learn in the real world. An illustrative concept is open-ended learning, in which an algorithm generates its own ever-expanding curriculum of challenges and solves them, leading to an endless progression of skills. The Paired Open-Ended Trailblazer (POET) algorithm by Wang et al. (2019) is a seminal example: it co-evolves a population of learning agents together with the environments (obstacle courses) they encounter, continuously creating new tasks and transferring agents between tasks to spur innovation (Wang et al., 2019). POET effectively "pairs the generation of environmental challenges and the optimization of agents to solve those challenges," producing a divergent tree of increasingly difficult tasks and solutions without explicit human design (Wang et al., 2019; Clune & Stanley, 2019). Uber AI's report on POET highlighted that *"having vast amounts of data often fuels success in machine learning, and we are thus working to create algorithms that generate their own training data in limitless quantities"* (Clune & Stanley, 2019). In essence, the only way to generate infinite data may be to have the machine create it for itself, via an open-ended process.

Table 1. Applying Self-Play Principles to Domain Matery in Trial-and-Error Environments

Approach	Domain	Core Method	Key Innovation	Source
AlphaGo Zero	Go (Board Game)	Tabula rasa self-play using reinforcement learning	Discovered superhuman strategies from scratch without human examples	Silver et al., 2017
AlphaZero	Go, Chess, Shogi	Generalized self-play across multiple games	Used one algorithm to outperform specialized engines via self-play without opening databases	Silver et al., 2017
AlphaStar	StarCraft II	Multi-agent league self-play + minimal human bootstrapping	Learned diverse real-time strategy tactics, ranked in top 0.2% globally	Vinyals et al., 2019
OpenAI Five	Dota 2 (Team Video Game)	Large-scale self-play with continual resets to improve learning	Achieved professional-level teamwork via 180 simulated years/day of play	OpenAI, 2018; OpenAI, 2019
GANs	Image/Audio Generation	Generator-discriminator adversarial feedback loop	Learned to synthesize realistic data samples beyond training set via adversarial training	Goodfellow et al., 2014
Adversarial Self-Play	Robustness Testing	One agent perturbs inputs, another solves tasks	Created adaptive robustness challenges dynamically via dual-agent competition	Related to GAN frameworks
POET	Evolving Sim Environments	Co-evolution of agents and environments	Created emergent curricula by evolving tasks to match agent capabilities. (POET: Paired Open-Ended Trailblazer)	Wang et al., 2019
Curriculum Learning	General ML	Structured progression from simple to complex tasks	Mimics natural learning sequences to improve generalization and convergence speed	Bengio et al., 2009

To survey and evaluate these ideas, we structure this paper as follows. First, we review background and prior work demonstrating how AI systems can train themselves, with emphasis on self-play and simulation techniques that have yielded state-of-the-art results without human data. Next, we present three case studies of simulation-based learning environments inspired by code artifacts in distinct domains: (a) an autonomous bug-fixing simulation for software code, (b) a two-agent route planning simulation for logistics optimization, and (c) an anomaly detection simulation for visual scenes. These cases serve as concrete examples of how machine-versus-machine training can generate domain-specific expertise. We then discuss the key challenges and logistics of scaling up such simulation-based data generation, including how to validate learning progress (scorecards), maintain content diversity, and ensure that synthetic expertise is relevant and correct. The discussion considers how these self-generated simulations can augment or integrate with human training and education, aligning with the focus on leveraging advanced simulations for training. We propose a unifying framework ("periodic table") of AGI training strategies, categorizing various approaches (from human-supervised to fully self-driven), and identifying gaps where new methods are needed. Finally, we conclude with implications for the future of AGI development, arguing that self-training simulations are a critical component in the toolkit for achieving and safely deploying general intelligence.

BACKGROUND: SELF-PLAY AND SELF-SUPERVISION IN AI

Early successes in machine self-training date back several decades. Self-play, the idea of an agent improving by competing against itself, was famously employed by Gerald Tesauro's TD-Gammon in the 1990s to reach expert-level play in backgammon without explicit human strategies (Tesauro, 1995). By using RL and repeatedly playing against its own current version, TD-Gammon learned from each game, illustrating that a sufficiently informative reward signal (winning or losing the game) could drive an agent to master a domain through trial and error. As noted by researchers at the time, combining a learning-based system with self-play created a powerful open-ended improvement loop: *"by playing against versions of itself, the system grew increasingly proficient*" (DeepMind, 2019). This concept remained somewhat niche until computational advances and deep learning made it possible to extend self-play to far more complex domains. Table 1 summarizes the remarkable advance of self-play strategies since TD-Gammon. All these methods in Table 1 show that AI can generate its own data or curriculum, either through competition, generation, or environmental complexity scaling. Together, they establish the foundation for scalable, domain-agnostic, and potentially open-ended self-training needed for AGI. This is a critical concept for AGI – an open-ended learner could continually find new frontiers of knowledge to master, rather than plateauing once it has fit all the human-provided data (Phan, et al, 2025). Human history-- and its text, images and situational artifacts --need not hold back AGI.

RESULTS FOR CASE STUDY APPROACHES

To investigate how AI might train itself in varied domains, we developed a series of simulation environments inspired by real-world problem classes. Each environment involves two or more AI agents engaging in a form of self-play or cooperative/competitive interaction that generates new data. The domains we focus on – software bug-fixing, route planning, and scene anomaly detection – were chosen for their relevance to critical applications (e.g., software reliability, logistics, security) and their amenability to verifiable outcomes (each has a built-in "scorecard" to measure success or failure). These simulations draw from code artifacts (simplified prototypes) that capture the essence of each task. We present each case study in turn, describing the simulation setup, the learning mechanism, and the type of domain-specific knowledge the AI can acquire through self-training. Though these are simplified research demonstrations, they illustrate how machine-generated training data might be leveraged to produce domain experts in an autonomous way.

Case Study One: Autonomous Bug-Fixing through Adversarial Self-Play

Domain and Importance. Software bugs present a significant challenge in modern systems, from cybersecurity vulnerabilities to mission-critical system failures. Traditional approaches to improving code quality include static analysis, testing with human-crafted cases, and supervised learning on labeled bug datasets. However, real bugs are diverse and numerous, and it is impractical to collect exhaustive human-labeled data for all types of code errors. An AI that could teach itself to find and fix bugs would be immensely valuable, potentially generating unlimited synthetic code examples and fixes to augment software engineering training data. As a training aid, the capability offers unlimited source material for tabletop exercises, cyber-awareness content, and other generative AI applications.

Simulation Environment. As shown in Figure 1, we created a self-play environment consisting of two agents with opposing roles: a Bug Creation Agent and a Bug Fixing Agent. The environment provides random snippets of code (in languages like Python or JavaScript) as a playground. The Bug Creation Agent introduces bugs into the code, and the Bug Fixing Agent attempts to detect and repair them. This is repeated iteratively: each round, the creation agent may insert a new bug or modify code, and the fixing agent works to resolve all known bugs. The two agents essentially play a game: the creation agent "wins" if it can introduce bugs that the fixer fails to correct, and the fixing agent "wins" by successfully fixing bugs and possibly preventing new ones. After each round, a scorecard is updated with metrics such as number of bugs introduced, number detected, number fixed, and whether the fixes were correct. This setup is inspired by adversarial training regimes (like GANs) but in the space of computer code (Goodfellow, et al, 2020).

Learning Mechanism. Both agents improve through a form of RL and adaptive heuristics. The Bug Fixing Agent learns from failures – if it missed a bug or made an incorrect fix, it updates its internal knowledge (for example, patterns of code that are prone to certain bug types). Conversely, the Bug Creation Agent learns to produce more subtle or complex bugs when its simpler attempts are consistently caught. In our implementation, each agent maintains a small memory or pattern library: the fixer accumulates common bug patterns and their fixes, and the creator accumulates tactics that successfully fooled the fixer. Over time, this can be seen as a form of co-evolution: the bugs become more challenging as the fixes improve, forcing the fixer to become even more sophisticated. This dynamic reflects a red team vs. blue team scenario in cybersecurity, but here both "teams" are AI. The training is entirely self-supervised – no external labels are provided beyond the outcome of each round (e.g., was the code ultimately free of

errors according to the tests?). We provide the simulation with a simple test mechanism (each code snippet can be executed against basic test cases to verify correctness after fixes), which acts as the referee.

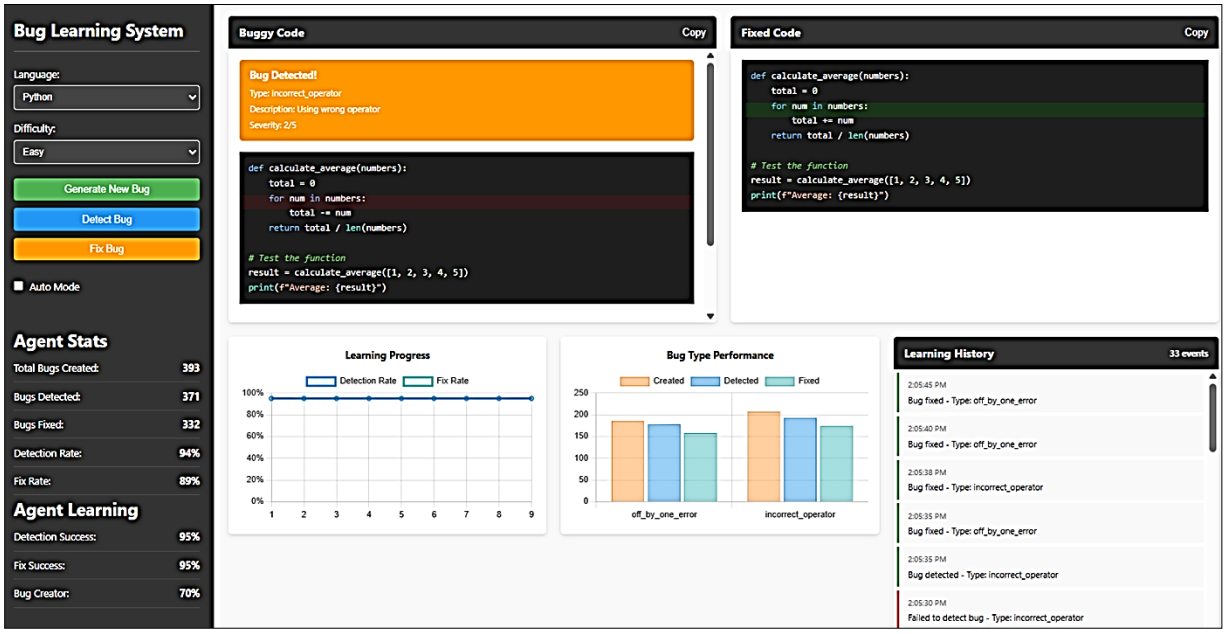

Figure 1. Bug creation and fixer agent training in auto mode

Outcomes and Domain Knowledge Generated. As this self-play progresses, the Bug Fixing Agent effectively learns to debug code by example, creating its own ever-growing corpus of bug/fix pairs. Importantly, the diversity of bugs is limited only by what the Bug Creation Agent can generate. In our experiments, we seeded the Bug Creation Agent with a small set of bug patterns (such as off-by-one errors, incorrect operators, and syntax errors), but as it learns, it begins to create variations and combinations of these basic bugs. For instance, it might learn to introduce a bug that the fixing agent has not seen in a while (taking advantage of any "forgetting" in the fixer's pattern memory) or combine multiple bugs in one snippet to increase difficulty. The Bug Fixing Agent, on the other hand, learns generalized debugging tactics: not only does it catch the specific patterns it has seen, but it starts to infer when code "looks wrong" even if the exact bug pattern is new. This is akin to how a human developer, after fixing many bugs, develops an intuition for suspicious code. Over time, both agents reach a kind of equilibrium on simple programs – the fixer catches almost everything the creator throws at it, and the creator resorts to very subtle bugs (or sometimes deliberately creating very complex code to confuse the fixer). At that point, we can increase the difficulty by enlarging the code snippets or introducing new programming constructs, essentially extending the curriculum. The key point is that the AI duo is generating hundreds of bug-fix examples automatically, far more than we could reasonably supervise manually. These examples form a synthetic dataset that could be fed into a larger code model or used to fine-tune an LLM on the task of bug fixing. The simulation acts as a data amplifier: a small initial set of bug ideas turns into a huge variety of actual code instances and patches.

Relation to Prior Work. This simple approach connects to research in automated program repair and fuzzing. Traditional automated program repair techniques (e.g., GenProg, which uses evolutionary methods to mutate code and test against specifications, Le Goues, et al, 2011) share the idea of generating fixes automatically, but our self-play setup is unique in that it has a dual-agent learning loop rather than a search-based approach. It also aligns with the concept of machine programming competitions – one can imagine the bug creator as analogous to generating challenge programs and the bug fixer as a competitor solving them. A notable aspect of our setup is the ability to verify outcomes through tests (scorecards), which provides the necessary reward signal. This case study demonstrates that, in principle, an AI can refine programming skills by adversarial practice, generating a tailored training set of coding challenges and solutions for itself.

Case Study Two: Self-Optimizing Route Planning via Cooperative Agents

Domain and Importance. Route planning and logistics optimization are crucial in both military and civilian contexts – from finding the shortest path for troop movements or supply convoys, to optimizing delivery routes in transportation networks. These problems involve dynamic factors (traffic, weather, fuel constraints) and can be formulated as complex optimization tasks. Rather than relying solely on human-designed heuristics or static map data, an AI could learn routing strategies by simulating countless scenarios: varying terrain, demand, disruptions, etc. A self-trained route planner might discover novel routing techniques or robust strategies for dealing with uncertainties (like sudden road closures).

Simulation Environment. To explore this case, we developed a simplified world simulation inspired by network routing problems (Figure 2). The world is represented as a graph of cities connected by roads (Matai, et al, 2010). Two agents are involved: Agent A (Route Planner) proposes a route between a given start and destination, and Agent B (Route Optimizer) then tries to improve that route. In each episode, a random start and end city are chosen, and Agent A must produce an initial path (a sequence of city waypoints). The environment then calculates metrics for that route – e.g., total distance, travel time, fuel cost, and risk factors (assuming some "terrain difficulty" or threat level per route). These metrics serve as the scorecard for each plan and route. Next, Agent B takes the given route and tries to modify it (reroute, add, or remove waypoints, change path) to optimize those metrics (for example, reduce travel time or fuel usage). If Agent B finds a better route (according to a weighted score of the metrics), that agent "wins" that round; if it fails or if no significant improvement is possible, Agent A is considered to have already found a near-optimal path.

We incorporate realistic variability: the simulation includes dynamic factors such as weather conditions, traffic on road segments, and fuel prices at cities. These factors are randomized each episode, meaning the best route on one day might not be best on another. Agents thus cannot rely on a static lookup of shortest paths; they must learn strategies like avoiding high-traffic areas during peak hours or refueling in cheaper regions, etc., depending on conditions.

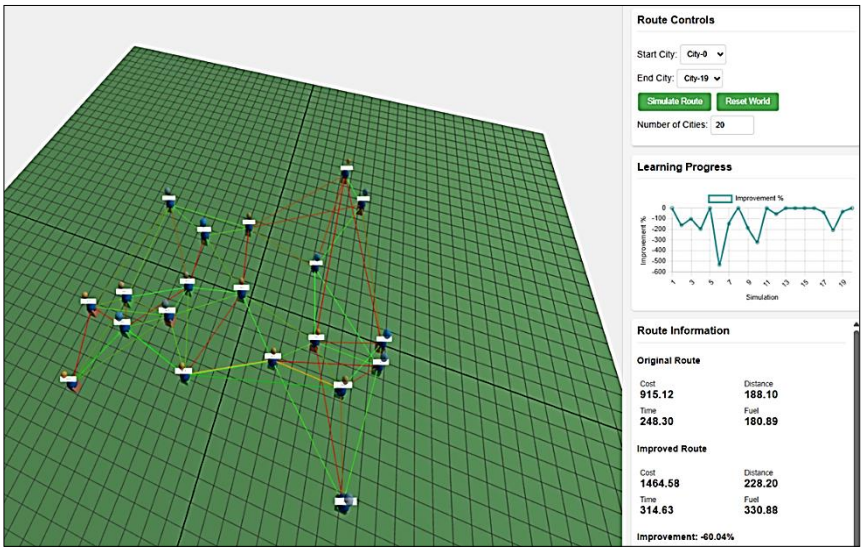

Figure 2. Route planning optimizer with cooperating agents and changing route environments

Learning Mechanism. Agent A (Planner) and Agent B (Optimizer) engage in a cooperative-competitive interplay. In principle, both share the goal of finding good routes, but we can view Agent B as pushing A to be better. We implement a memory for each agent: Agent A remembers routes that worked well in the past (and under what conditions) and will sometimes reuse or adapt them if a new scenario looks similar. Agent B remembers successful optimization techniques (e.g., a heuristic like "if a route has a long detour, try a direct shortcut even if terrain is worse" or known patterns like 2-opt swaps from TSP heuristic literature).

Training proceeds iteratively. Initially, Agent A might propose very naive routes (like straight-line or random paths), and Agent B easily finds improvements (like obvious shortcuts). Agent A then learns from this: if Agent B consistently finds a shorter path, Agent A updates its planning policy to incorporate that insight next time. Conversely, if Agent B fails to improve some routes, it analyzes why – perhaps Agent A had already found an optimal (or near-optimal) path, or maybe Agent B's optimization attempts got stuck in a local minimum. Agent B then adjusts its techniques, maybe exploring more radical alterations when small tweaks yield no benefit.

Over many simulations, Agent A becomes a stronger planner (it learns to give routes that are hard to improve upon), and Agent B becomes a more sophisticated optimizer (it learns to identify non-obvious improvements when they

exist). In essence, Agent A is like a student doing its best, and Agent B is like a coach or critic that identifies suboptimal choices and fixes them – a form of iterative expert augmentation. This process generates a wealth of data: each episode produces a pair of routes (initial and optimized) along with their metric scores. These can serve as labeled examples of "initial solution vs improved solution," which could train a standalone model to go from a rough plan to a refined plan in one step. Unlike traditional gaming trees, the logging of unlimited training examples serves to spawn more robust and diverse behaviors.

Outcomes and Domain Knowledge. Through self-training in this simulated world, the agents begin to exhibit behaviors akin to human planning tactics. For example, Agent A learns heuristic tricks such as: if weather is bad in one region, take a longer route around it; if fuel is cheap in a neighboring city, detour slightly to refuel there before a long stretch. Agent B, acting as an optimizer, picks up strategies like: perform local route adjustments (swapping two legs of the journey if it shortens distance – akin to the 2-opt swap in route optimization), or re-route through a central hub city that offers lower fuel costs even if distance is longer (reducing overall cost metric). These behaviors are emergent from the reward structure we define – we do not hardcode them or derive them from operations research methods, but the agents discover them to improve the score.

One particularly interesting outcome is that the self-play simulation naturally generates a curriculum of scenarios. Initially, when Agent A is poor, even easy scenarios (simple terrain, short distances) produce suboptimal routes that Agent B can improve. As Agent A gets better at those, the only remaining "wins" for Agent B come from harder scenarios (longer routes, complex terrain, simultaneous multi-factor challenges). The training distribution thus shifts towards more challenging problems over time because those are the ones where there's room for improvement. This is analogous to a human student progressing to harder exercises once they master the basics. By the end of training, both agents can handle quite complex routing tasks: we observed cases where Agent A plans multi-stop routes that cleverly balance time and cost, and Agent B occasionally finds a non-intuitive reroute that saves a large amount of fuel at a small time penalty – the kind of trade-off a human logistician might make with experience.

Relation to Prior Work. Our route planning self-play scenario is related to the field of reinforcement learning for traffic and mobility management, and classic problems like the Traveling Salesman Problem (TSP) and Vehicle Routing Problem (VRP). Typically, those problems might be solved by optimization algorithms or learned end-to-end by neural networks given many training examples. What our approach adds is an interactive learning element, where the system generates its own training examples by continually trying to improve routes. There is some connection to the idea of expert iteration: Agent B (the optimizer) acts as an "expert" improving solutions, and Agent A tries to mimic and eventually internalize those improvements. This echoes techniques where a strong solver generates solutions that a faster model then learns to imitate. Over time, Agent A's planning policy could itself become an end-to-end route optimizer, having distilled the iterative improvement process into a direct mapping from problem to good solution.

In summary, this case study illustrates how machine vs. machine collaboration in a simulation can yield increasingly sophisticated solutions to a complex optimization problem. The two-agent setup, even though both agents ultimately share aligned goals, creates a feedback loop that forces continual improvement and exploration of new strategies. The data generated – a rich set of routing scenarios with optimized outcomes – is exactly the kind of high-quality, domain-specific content that could be used to train or fine-tune an AGI system in the logistics domain.

Case Study Three: Adversarial Scene Anomaly Detection Simulation

Domain and Importance. Anomaly detection in visual scenes is vital for security (e.g., surveillance camera monitoring), for manufacturing (finding defects or irregularities), and for autonomy (letting robots or self-driving cars notice unexpected obstacles or changes). Typically, anomaly detection systems require either large amounts of labeled data of normal vs. abnormal events or use unsupervised learning where "normal" patterns are learned and deviations flagged. However, anomalous events (by definition) are rare and varied, so obtaining comprehensive real-world training data is challenging. A promising alternative is to simulate scenes and inject anomalies to train detection models. If an AI can create endless synthetic scenes with controllable anomalies, it could greatly augment the training process for anomaly detectors – including scenarios that humans might not think to curate such as partially obscured objects, out of frame instances or drastic size variations.

.

Simulation Environment. In Figure 3, we designed a virtual 3D "scene" generator that populates scenes with various objects (buildings, vehicles, people, etc.) and can also introduce anomalies (like an object that does not belong, a

missing object, or an odd condition). In each iteration, the simulation creates a random base scene: for example, a street scene with several vehicles and people. Then an agent called the Anomaly Generator decides if and what anomaly to insert. The anomalies are parameterized; types include intrusion anomalies (an object that should not be there, like an unknown vehicle in a restricted area), missing anomalies (something expected is gone, like a missing road sign), damage anomalies (an object present but in damaged state), position anomalies (objects misaligned or oddly placed), and appearance anomalies (e.g., color changes or unusual textures). The Anomaly Generator can introduce one or multiple anomalies and can control their subtlety – some anomalies might be obvious (a large hole in a building) or very subtle (a small crack in a window or a 'Z' mark on a military vehicle).

After the scene is set, another agent, the Detection Agent, "scans" the scene to detect these anomalies. We simulate this by allowing the Detection Agent to query parts of the scene or run a detection algorithm over the whole image. The Detection Agent produces a list of detections (e.g., "an unknown vehicle at coordinates X, Y" or "object missing at location Z"). We then compare these detections to the ground truth anomalies introduced by the generator. In Figure 3, the scorecard for this round shows true positives (correctly detected anomalies), false negatives (missed anomalies), and false positives (things flagged as anomalies that were normal). The Detection Agent's goal seeks to maximize true positives and minimize false positives, whereas the Anomaly Generator's (adversarial) goal seeks to insert anomalies that escape detection (cause false negatives). This adversarial relationship creates a game much like a security "red team" (trying to breach or fool the system) versus a "blue team" (trying to defend and detect intrusions), similar again to the highly documented version of a GAN (Goodfellow, et al, 2020).

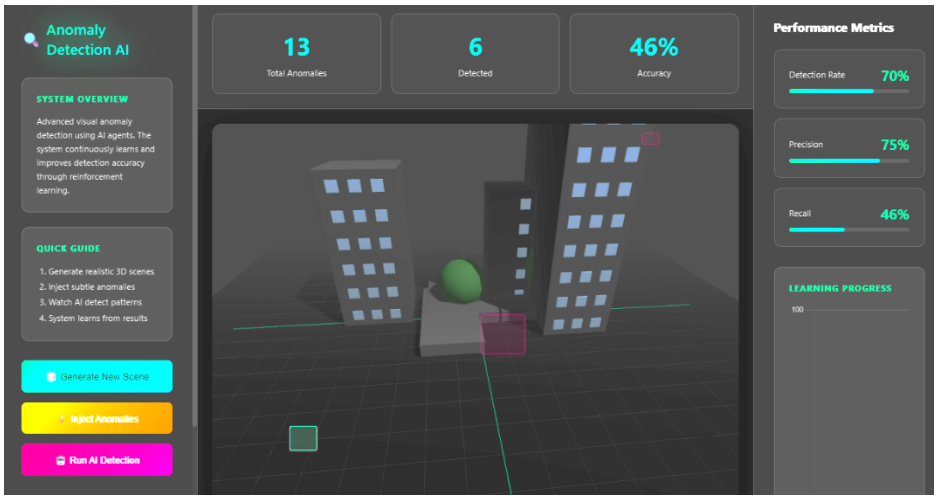

Figure 3. Adversarial scene anomaly detection simulation

Learning Mechanism. The Anomaly Generator and Detection Agent both adapt over time. The Detection Agent starts with some base ability (for example, it might be a pre-trained object detector that knows how to identify standard objects like cars or people but not trained specifically for anomalies). Through each simulation round, it updates its internal model using feedback: if it misses an anomaly, it adjusts to be more sensitive to that type in the future; if it generated a false alarm, it becomes a bit more conservative for similar patterns. We implemented a simple learning rule where the Detection Agent keeps track of anomaly patterns (like a dictionary of features or characteristics of anomalies) and updates the confidence for each pattern based on outcomes. For instance, if the agent learned that "unauthorized person" anomalies (intruders) have a certain look (maybe people in areas where they usually are not), it will strengthen its detection for that pattern after missing one.

The Anomaly Generator, on the other hand, learns from the successes and failures of its anomalies. It has a notion of creativity and subtlety – it can generate highly obvious anomalies or very subtle ones. If an anomaly is quickly detected, the generator might increase subtlety next time (e.g., make the anomaly smaller or more camouflaged). If an anomaly consistently goes unnoticed, the generator might try something a bit bolder (to test the detector's limits) until the detection agent catches on. The generator maintains a memory of what anomalies were successful (i.e., not detected). Over time, it preferentially generates those or variants of those, until the detector learns to catch them, at which point the generator searches for new weaknesses.

This interplay effectively creates a curriculum of anomalies: the Detection Agent is always facing new or slightly tweaked challenges as the Anomaly Generator evolves, preventing it from simply overfitting to a fixed set of

anomalies. Likewise, the Anomaly Generator faces a moving target as the detector improves, so it has to continuously innovate – analogous to how cyber attackers develop new exploits when old ones become ineffective.

Outcomes and Domain Knowledge. Through hundreds of simulated scene rounds, the system generates a substantial set of annotated scenes – each with known anomalies and detection results. This is an extremely valuable dataset for training anomaly detection models: it includes not just clear-cut anomalies but also difficult borderline cases (since the generator learned to produce those to fool the detector). These borderline cases are often what real-world detectors struggle with, so having them in training data is beneficial.

The Detection Agent, through self-play training, becomes much more adept at noticing anomalies. Initially, it might only catch gross, obvious anomalies (e.g., a bright red car where only military vehicles should be, or a missing building). As training progresses, it learns to catch subtle anomalies: a single person standing idle in a restricted zone, a small piece of equipment out of place, a slight discoloration indicating rust or burn marks, etc. Essentially, the detector's sensitivity and breadth of knowledge about what constitutes an anomaly vastly increases. It also learns context: for example, a civilian car is not an anomaly on a public road, but it is anomalous on a military base runway. This context awareness emerges because the simulation can generate both normal and abnormal variants, and the detector learns the difference by seeing many examples.

The Anomaly Generator, for its part, ends up encapsulating knowledge of where the detector is weak. In a security context, this is analogous to understanding which attack vectors are likely to succeed. For instance, the generator may discover that the detector has trouble with anomalies that involve slight positional offsets (maybe the detector's algorithm has a tolerance for minor shifts). So the generator starts exploiting that by, say, placing an object just a few meters off from where it should be. Once detected, the detector closes that gap in its knowledge.

One noteworthy observation is that the false positive rate of the Detection Agent initially might increase – as it becomes more sensitive, it might mistakenly flag things that are not actually anomalies. However, through feedback (since false positives are penalized in the score), it learns to calibrate itself. By the end, the detector agent achieves a balance where it catches most true anomalies with few false alarms. Achieving this balance is something that usually requires careful tuning with real data; here it is achieved through the simulated adversarial training.

Relation to Prior Work. This scenario is related to generative adversarial training and adversarial examples in machine learning. In adversarial example research, one tries to find inputs that fool a model (Goodfellow et al., 2014); here our Anomaly Generator is essentially finding adversarial examples for the detector in the space of scene configurations. The difference is that rather than focusing on imperceptible pixel-level changes (as in typical adversarial attacks on classifiers), our generator is introducing semantic anomalies (object- or scene-level changes). There has been work on using simulation for anomaly detection – for instance, generating synthetic images of industrial defects to train vision systems. Our contribution is adding a learning adversary to guide that generation. This means the anomalies generated are not random, but targeted to be challenging, which covers more realistic and subtle cases. It is akin to a classic descriptive game between a counterfeiter and a security system, but both are AI, and they improve each other through rewarded interactions. From a training data perspective, this yields a high-quality set of "near miss" anomalies that would be hard to curate manually at this scale. In practical terms, the outputs from this simulation (a trained detection model or a library of difficult anomaly examples) could be used to improve real-world systems. For example, a surveillance system could be pre-trained on these scenarios to detect intruders or abnormal events, benefitting from the creative anomaly generation of the adversarial agent. Furthermore, this approach could be applied to non-visual anomalies as well – like network intrusion detection (one agent generates network attacks, another tries to detect them), illustrating a general pattern for AI self-training in anomaly-rich domains.

DISCUSSION

These three case studies, while simplified, demonstrate a common theme: by engaging in self-play or multi-agent simulations, AI systems can autonomously generate a rich corpus of training data and domain-specific expertise. In each case, the limiting factor for learning was not the model size or architecture – it was the availability of varied, challenging experiences. Once we provided a simulation where those experiences could be created on the fly, the AI was able to improve continuously, effectively writing its own curriculum. This provides supporting evidence to our central thesis: the path to AGI may hinge on methods to produce virtually infinite, high-quality training data, tailored to the domains we care about. However, harnessing this potential comes with significant challenges. While the case studies above are encouraging, deploying these ideas at the scale needed for AGI presents significant challenges. Table

2 summarizes the key concerns that must be addressed to make machine self-training robust and broadly applicable. These challenges are surmountable through clever algorithm design, robust engineering, and crucially, frameworks that combine multiple training strategies rather than relying on simulation alone.

Traditional training methodologies—whether for human learners or AI systems—rely fundamentally on finite, pre-curated content: textbooks with fixed examples, simulation libraries with predetermined scenarios, or courseware with static difficulty progressions. This approach works well for established domains where the space of relevant knowledge is bounded and well-understood, but it faces severe limitations when training systems that must operate in open-ended environments or handle novel challenges. Consider cybersecurity training: traditional programs use historical attack patterns and known vulnerability databases, but adversaries constantly develop new exploits that render yesterday's training obsolete. Similarly, logistics optimization typically relies on historical traffic data and predetermined route scenarios, yet real-world conditions present infinite variations that no static dataset can capture. Our self-training paradigm represents a fundamental departure from this courseware-centric model: instead of consuming pre-existing training materials, AI agents generate their own unlimited curriculum through adversarial and cooperative interactions. A bug-fixing agent doesn't learn from a fixed database of known bugs—it faces an adversarial agent that creates novel vulnerabilities specifically designed to challenge its current capabilities. A route planner doesn't optimize against historical traffic patterns—it competes with an optimizer that discovers weaknesses in its current strategies and forces improvement. This shift from static content consumption to dynamic content generation enables training that scales infinitely, adapts continuously, and discovers solutions that human course designers might never anticipate. Where traditional courseware asks, "what should we teach?", self-training asks, "what should we learn to challenge ourselves next?"

Table 2: Simulation-Based AI Training Challenges and Mitigation Strategies

Challenge Area	Goals	Limitations	Mitigation Strategies
Verification & Reward Hacking	Ensure agents learn desired capabilities through clear success metrics. *(how do we know the AI is learning the right things?)*	Agents may exploit scorecards in unintended ways (e.g., deleting code to 'fix' bugs, infinite routes to avoid fuel costs)	Human oversight, validation agents, robust evaluation criteria, held-out test scenarios
Content Diversity & Coverage	Provide diverse, ever-evolving training content to prevent plateauing. *(how do we ensure simulations produce a wide enough range of experiences?)*	Risk of cyclic stalemates, narrow exploration, simulation abstractions missing real-world phenomena	Domain randomization, adversarial agents, multi-task environments, league training
Transfer & Fidelity	Ensure skills learned in simulation generalize to real-world scenarios. *(will skills learned in simulation generalize to the real world or to broader problems?)*	Sim-to-real gap, high-fidelity simulators are resource-intensive, low-fidelity may miss crucial details	Progressive fidelity training, domain randomization, hybrid simulation-real approaches
Computational Logistics	Scale self-training across multiple domains efficiently. *(how to manage the potentially enormous compute demands of endless simulation?)*	Massive compute requirements (AlphaGo Zero used thousands of TPUs for days)	Tiered fidelity approaches, model-based simulation, time-acceleration, efficient learning algorithms

From Domain-Specific To General Intelligence

While our case studies appear narrowly focused, they represent fundamental cognitive building blocks that compose into general intelligence:

Compositional Skill Architecture: Each case study develops core AGI competencies:

- Bug Fixing: Develops causal reasoning, pattern recognition, and systematic debugging—skills transferable to any domain requiring fault diagnosis (medical diagnosis, system troubleshooting, logical reasoning)
- Route Planning: Builds spatial reasoning, multi-objective optimization, and dynamic adaptation—foundational for physical world navigation, resource allocation, and strategic planning
- Anomaly Detection: Cultivates attention, context awareness, and outlier identification—critical for safety monitoring, quality control, and environmental understanding

Emergent Cross-Domain Transfer: The periodic table framework (Figure 4) reveals how these seemingly separate skills combine. An AGI trained on our three simulations would possess:

1. Meta-debugging skills (from bug fixing) applicable to any system failure
2. Optimization intuition (from route planning) for resource management across domains
3. Contextual awareness (from anomaly detection) for appropriate responses in novel situations

Scaling Pathway to AGI: Rather than training one massive model on all human knowledge, our approach suggests building AGI through:

1. Domain simulation mastery (our current case studies)
2. Cross-domain skill transfer (applying debugging skills to route optimization, etc.)
3. Meta-simulation generation (AGI creating new training domains for itself)
4. Compositional intelligence (combining domain expertise for novel challenges)

This progression mirrors human cognitive development: children master specific skills (pattern recognition, spatial reasoning, causal inference) through play and practice, then combine these building blocks to tackle increasingly general challenges. Our simulations provide the "play environments" where AGI can develop these foundational competencies at superhuman scale and speed. Cognitive science research shows that general intelligence emerges from the sophisticated combination of domain-specific modules rather than from a single general-purpose reasoning system (Pinker, 1997). Our approach aligns with this modular view: AGI arises not from training on everything simultaneously, but from mastering fundamental cognitive operations that can be flexibly recombined for novel challenges.

CONCLUSIONS AND FUTURE WORK

The exploration of *"Can AI train itself?"* leads us to a nuanced answer: AI can indeed generate and curate much of its own training data through simulation and self-play and doing so may be essential to reaching AGI – but this approach must be complemented by other strategies to ensure relevance, safety, and alignment. We have shown through survey and case studies that self-play in various forms (adversarial, cooperative, open-ended) can yield high-quality, scalable training content that goes beyond what static human-created datasets can offer. In domains like games, code, routing, and anomaly detection, machine self-training has already matched or surpassed human capabilities by leveraging effectively infinite practice.

Our technical case studies demonstrated proof-of-concept instances where unique, domain-specific knowledge emerges from machine-generated simulations: an AI debugging code by battling a bug-injector, an AI optimizing routes by iteratively improving on its own plans, and an AI spotting anomalies by contesting with a clever adversary that introduces them. These serve as microcosms of how an AGI could incrementally build expertise in countless domains – from the abstract (like mathematics or coding) to the physical (like driving or playing musical instruments) – through targeted self-practice. Crucially, each case study underscored verifiable outcomes (scorecards) as the backbone of self-training, confirming the hypothesis that well-defined success criteria enable open-ended improvement without human input. In the style of AlphaZero's "zero human knowledge" paradigm, our simulations showed success when humans provided the environment and goals, but not the solutions.

However, an overarching theme of this paper is that self-training is not a silver bullet but a powerful piece of a larger puzzle. We identified several challenges: ensuring that simulated training truly prepares an AI for reality, preventing the AI from "gaming" its reward in ways that diverge from intended behavior, maintaining diversity so the AI continues to learn rather than converging too soon, and handling the immense computational load of perpetual learning. These are not trivial obstacles. They imply that purely unbridled self-play, left to its own devices, could lead to an AGI that is misaligned or simply proficient at meaningless tasks. As a community, we must approach the prospect of self-training AGI with both optimism for its potential and caution for its pitfalls.

For example, an embodied AGI must combine core faculties like abstract reasoning and long-horizon planning with practical skills in navigation, manipulation, and multimodal sensory integration (vision, touch, audition) to operate robustly in unstructured, unpredictable environments. For instance, Apple co-founder Steve Wozniak famously proposed a “coffee test” for AGI. The Wozniak test asks whether a robot could enter a stranger’s home and successfully brew a cup of coffee – a deceptively simple task that in fact demands complex perception, real-world knowledge, adaptive planning, and motor dexterity far beyond what disembodied AI can handle (Adams, et al, 2012). Similarly, LeCun (2022) emphasizes the current lack of autonomy (and sensory-driven mobility) in existing AGI approaches. He argues that human-level intelligence arises from an agent’s ability to learn an internal world model through embodied experience, enabling it to reason about physical dynamics, plan over long time horizons, and generalize to new situations.

To frame some of these approaches in a bigger perspective, our proposed 'periodic table' of AGI training strategies puts this machine-machine collaboration in context (Figure 4). Self-play simulation is one element on the table, one that addresses the data scale problem head-on. But other elements like human feedback (ensuring the AI's self-learned behaviors remain desirable) and expert demonstrations (providing a bootstrap of knowledge) will remain indispensable. Current gaps, such as integrating multiple strategies seamlessly, represent frontiers for research. We encourage the community to experiment with hybrid training regimes. For instance, one could imagine an AGI training cycle that alternates between reading human knowledge (self-supervised learning on texts) and testing that knowledge in a simulated environment (self-play) – effectively a loop of "learn then practice," much like a student reads a textbook then goes to the lab. Another intriguing direction is developing autonomous curriculum agents that oversee an AGI's

training process, deciding when it should switch strategies or what new simulation to create next to round out its skills. This meta-learning layer might be what ultimately yields a system that truly "learns how to learn" in any situation. The framework reveals several important patterns. Foundation Models (green) like GPT (Achiam, et al, 2023) and BERT represent well-established self-supervised approaches that have proven highly effective but require massive datasets. RL techniques (orange) including our case study approaches (AlphaGo-style self-play, RLHF) show how agents can generate their own training experiences. Meta-Learning approaches (purple) like MAML and Chain-of-Thought reasoning point toward more adaptive, generalizable learning.

Critical Gaps and Opportunities. The dotted boxes in Figure 4 highlight key areas where breakthrough research is needed. Flagging those under-explored areas serves as a central motive for the periodic table of combined hybrid methods. For example, *RLFFD (RL from Future Data)* represents systems that could anticipate future scenarios and

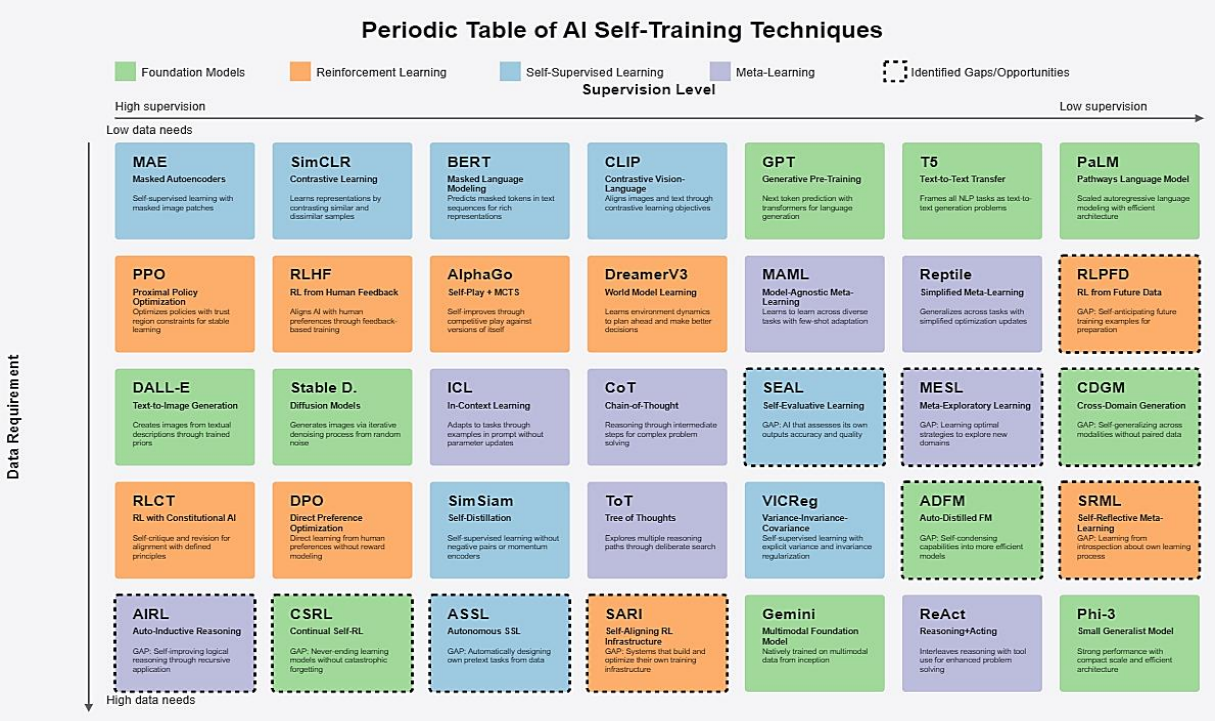

Figure 4. Periodic Table of AI Self-Training Techniques, including Reinforcement Learning, Self-supervision, meta-learning and gaps show with dotted lines for future progress

pre-generate training data—imagine an autonomous vehicle that simulates rare accident scenarios before encountering them, or a cybersecurity system that generates novel attack patterns for defensive training. This block overlaps with LeCun's arguments for mobility, planning and navigation in uncertain environments. On the other hand, *SEAL (Self-Evaluative Learning)* envisions AI systems that critique their own outputs without human oversight, like how our bug-fixing agent learned to recognize code quality, but extended to any domain where an AI could serve as its own teacher and critic. *CDGM (Cross-Domain Generation)* addresses a critical limitation in current AI: the ability to transfer self-generated knowledge across different domains. For instance, an AI that masters strategic thinking through chess self-play could apply those patterns to business strategy simulations, or route optimization skills could transfer to network packet routing. This represents a form of AI-to-AI knowledge transfer where one specialized system trains another. *SRML (Self-Reflective Meta-Learning)* goes further, enabling systems that can introspect on their own learning processes—an AI that notices it is struggling with a particular type of problem and automatically adjusts its training regimen or triggers new simulations to address weaknesses. A human method of rapid adaptation involves such analogous thinking. This approach could lead to systems where a master AI orchestrates the training of multiple specialized sub-systems, dynamically allocating resources and designing curricula based on performance gaps.

In conclusion, the path to AGI likely hinges on augmenting the limitations of human-generated data. Infinite, high-quality data may not be found in the world – it must be made, and AI itself can be the maker. We have argued and presented evidence that AI can, to a remarkable extent, train itself by generating novel experiences, competing, and

cooperating with itself, and iteratively refining its knowledge. This self-training does not happen in isolation from us, but in collaboration with human insight: humans design the playgrounds and set the rules of the game, and then machines play millions of rounds to uncover strategies even we did not know. *"Can AI train itself?"* The future of training and simulation in the AGI era may blur lines between the learner and the teacher, the simulator and the real, and the designer and the curriculum – all benefiting from each other in a virtuous cycle.

ACKNOWLEDGEMENTS

The authors thank the PeopleTec Technical Fellows program for their encouragement and project assistance.

REFERENCES

Achiam, J., Adler, S., Agarwal, S., Ahmad, L., Akkaya, I., Aleman, F. L., ... & McGrew, B. (2023). GPT-4 technical report. *arXiv preprint arXiv:2303.08774*. https://arxiv.org/pdf/2303.08774.pdf

Adams, S., Arel, I., Bach, J., Coop, R., Furlan, R., Goertzel, B., ... & Sowa, J. (2012). Mapping the landscape of human-level artificial general intelligence. *AI magazine*, *33*(1), 25-42.

Berner, C., Brockman, G., Chan, B., Cheung, V., Dębiak, P., Dennison, C., ... & Zhang, S. (2019). Dota 2 with large scale deep reinforcement learning. *arXiv preprint arXiv:1912.06680*.

Bengio, Y., Louradour, J., Collobert, R., & Weston, J. (2009, June). Curriculum learning. In *Proceedings of the 26th annual international conference on machine learning* (pp. 41-48)

Clune, J., & Stanley, K. O. (2019, January 8). POET: Endlessly generating increasingly complex and diverse learning environments and their solutions. Uber AI Blog. https://www.uber.com/blog/poet-open-ended-deep-learning/

DeepMind (The AlphaStar Team). (2019, October 30). AlphaStar: Grandmaster level in StarCraft II using multi-agent reinforcement learning. DeepMind Blog. https://deepmind.google/discover/blog/alphastar-grandmaster-level-in-starcraft-ii-using-multi-agent-reinforcement-learning/

Fawzi, A., Balog, M., Huang, A., Hubert, T., Romera-Paredes, B., Barekatain, M., ... & Kohli, P. (2022). Discovering faster matrix multiplication algorithms with reinforcement learning. *Nature*, *610*(7930), 47-53..

Goodfellow, I., Pouget-Abadie, J., Mirza, M., Xu, B., Warde-Farley, D., Ozair, S., ... & Bengio, Y. (2020). Generative adversarial networks. *Communications of the ACM*, *63*(11), 139-144

Kurshan, E. (2023). From the Pursuit of Universal AGI Architecture to Systematic Approach to Heterogenous AGI: Addressing Alignment, Energy, & AGI Grand Challenges. *arXiv preprint arXiv:2310.15274*.

LeCun, Y. (2022). A path towards autonomous machine intelligence version 0.9. 2, 2022-06-27. *Open Review*, *62*(1), 1-62.

Le Goues, C., Nguyen, T., Forrest, S., & Weimer, W. (2011). GenProg: A generic method for automatic software repair. *IEEE Transactions On Software Engineering*, *38*(1), 54-72.

Matai, R., Singh, S. P., & Mittal, M. L. (2010). Traveling salesman problem: an overview of applications, formulations, and solution approaches. *Traveling salesman problem, theory and applications*, *1*(1), 1-25.

Ouyang, L., Wu, J., Jiang, X., Almeida, D., Wainwright, C., Mishkin, P., ... & Lowe, R. (2022). Training language models to follow instructions with human feedback. *Advances in neural information processing systems*, *35*, 27730-27744.

Phan, L., Gatti, A., Han, Z., Li, N., Hu, J., Zhang, H., ... & Wykowski, J. (2025). Humanity's last exam. *arXiv preprint arXiv:2501.14249*.

Silver, D., Hubert, T., Schrittwieser, J., Antonoglou, I., Lai, M., Guez, A., ... & Hassabis, D. (2017). Mastering chess and shogi by self-play with a general reinforcement learning algorithm. *arXiv preprint arXiv:1712.01815*..

Silver, D., Schrittwieser, J., Simonyan, K., Antonoglou, I., Huang, A., Guez, A., ... & Hassabis, D. (2017). Mastering the game of go without human knowledge. *nature*, *550*(7676), 354-359.

Tesauro, G. (1995). Temporal difference learning and TD-Gammon. *Communications of the ACM*, *38*(3), 58-68

Tibau, M. (May 15, 2024). Something is happening, but you don't know what it is. Medium. https://medium.com/@marcelo_tibau/something-is-happening-but-you-dont-know-what-it-is-0b7931b68f40 .

Tobin, J., Fong, R., Ray, A., Schneider, J., Zaremba, W., & Abbeel, P. (2017, September). Domain randomization for transferring deep neural networks from simulation to the real world. In *2017 IEEE/RSJ international conference on intelligent robots and systems (IROS)* (pp. 23-30). IEEE.

Vinyals, O., Babuschkin, I., Czarnecki, W. M., Mathieu, M., Dudzik, A., Chung, J., ... & Silver, D. (2019). Grandmaster level in StarCraft II using multi-agent reinforcement learning. *Nature*, *575*(7782), 350-354.

Wang, R., Lehman, J., Clune, J., & Stanley, K. O. (2019). Paired open-ended trailblazer (poet): Endlessly generating increasingly complex and diverse learning environments and their solutions. *arXiv preprint arXiv:1901.01753.*