Instituto Tecnológico y de Estudios Superiores de Occidente

Reconocimiento de validez oficial de estudios de nivel superior según acuerdo secretarial 15018, publicado en el Diario Oficial de la Federación del 29 de noviembre de 1976.

Department of Mathematics and Physics

Master of Data Science

Regularization of visual transformers and reinforcement learning: an approach to general artificial intelligence evaluated through the ARC benchmark

---

THESIS to obtain the **DEGREE** of
MASTER OF DATA SCIENCE

A thesis presented by:
Mario Alejandro Oviedo Vázquez

Thesis Advisors:
Dr. Edgar Alejandro Guerrero Arroyo

Tlaquepaque, Jalisco, June, 2025

Instituto Tecnológico y de Estudios Superiores de Occidente

Reconocimiento de validez oficial de estudios de nivel superior según acuerdo secretarial 15018, publicado en el Diario Oficial de la Federación del 29 de noviembre de 1976.

Department of Mathematics and Physics

Master of Data Science Approval Form

Thesis Title: **Regularization of visual transformers and reinforcement learning: an approach to general artificial intelligence evaluated through the ARC benchmark**

Author: **Mario Alejandro Oviedo Vázquez**

Thesis Approved to complete all degree requirements for the Master of Science Degree in Data Science.

Thesis Advisor, **Dr. Edgar Alejandro Guerrero Arroyo**

Thesis Reader, **Dr. Fernando Ignacio Becerra Lopez**

Thesis Reader, **Dr. Riemann Ruíz Cruz**

Academic Advisor, **Dra. Rocío Carrasco Navarro**

Tlaquepaque, Jalisco, June, 2025

Regularization of visual transformers and reinforcement learning: an approach to general artificial intelligence evaluated through the ARC benchmark

Mario Alejandro Oviedo Vázquez

Abstract

With the paradigm shift introduced by the Transformer architecture and autoregressive language models, recent years have witnessed an increased implementation of these models across numerous fields of human endeavor. The rise in their usage has brought forth a new set of challenges stemming from inherent limitations and deficiencies within these models. Specifically, autoregressive language models continue to exhibit limitations when confronted with simple problems that were not part of their training data, demonstrating either a lack or extremely limited skills in comprehension, reasoning, and planning.

These cognitive skills are particularly valuable for real-world applications where edge cases and scenarios not previously encountered during training occur frequently and require accurate evaluation. For instance, in self-driving cars, camera input glitches or altered dynamics that were absent from the training dataset are critical scenarios that require precise and reliable assessment.

This thesis introduces a modified encoder visual transformer (ViT) with reinforcement learning (RL) to try to better address the current problems presented by autoregressive models. To evaluate the performance of this proposal and compare it with other possible solutions, The Abstraction and Reasoning Corpus (ARC) is used, which is a benchmark designed to evaluate these qualities in AI models. It was developed in 2019, and unlike other benchmarks in which models manage to surpass human performance in a relatively short period, the ARC benchmark has not been able to cross that threshold in the last 5 years.

A new approach to address the reasoning challenges of current models, one example has been a simple transformer which tends to overweight vector embeddings (VE) over positional encoding (PE).This tendency leads to the model overlooking crucial positional information, which often provides essential criteria for solving the problem, especially under certain scenarios. introducing a learnable regularization factor between the model can compensate for this overweight. Other major challenge is handling the out-of-training data, previous investigations solved this challenge in closed space problems, the model try to solve this using Q-Learning a Reinforce Learning technique where the model gets rewards as it finds the best policy from an action space, in this scenario there is not space restriction for possible solution; however, by restricting the actions the model can perform the training push to converge in a solution that can generalize well.

Keywords: Reinforcement Learning, Artificial Intelligence, Transformers, ARC Benchmark.

Regularización de un Visual Transformer y Aprendizaje por Refuerzo: Un acercamiento a la Inteligencia Artificial General evaluado a través del ARC Benchmark

Mario Alejandro Oviedo Vázquez

Resumen

Con el cambio de paradigma introducido por la arquitectura Transformer y los modelos de lenguaje autoregresivos, en los últimos años se ha observado una creciente implementación de estos modelos en numerosos campos de la actividad humana. Este aumento en su uso ha traído consigo un nuevo conjunto de desafíos derivados de limitaciones y deficiencias inherentes a dichos modelos. En particular, los modelos de lenguaje autoregresivos siguen mostrando limitaciones cuando se enfrentan a problemas simples que no formaron parte de sus datos de entrenamiento, evidenciando una falta de habilidades en comprensión, razonamiento y planificación.

Estas habilidades cognitivas son especialmente valiosas para aplicaciones del mundo real, donde los casos límite y los escenarios no contemplados durante el entrenamiento ocurren con frecuencia y requieren una evaluación precisa. Por ejemplo, en los autos autónomos, un fallo en la entrada de la cámara o alteraciones en la dinámica de manejo que no estuvieron presentes en el conjunto de datos de entrenamiento son escenarios críticos que exigen una evaluación confiable y exacta.

La tesis introduce un transformer con codificador visual modificado (ViT) y aprendizaje por refuerzo (RL). La intencion es intentar abordar de mejor manera los problemas que presentan los modelos autoregresivos. Para evaluar el rendimiento de esta propuesta y compararla con otras soluciones posibles, se utiliza el Abstraction and Reasoning Corpus (ARC), un benchmark diseñado para evaluar estas cualidades en modelos de inteligencia artificial. Fue desarrollado en 2019 y, a diferencia de otros benchmarks en los que los modelos logran superar el desempeño humano en un periodo relativamente corto, el benchmark ARC no ha podido superar ese umbral en los últimos cinco años.

El Transformer Visual Regularizado con Aprendizaje por Refuerzo (ViT-RL). Este modelo revela que los transformers tienden a sobrevalorar los vectores de embedding (VE) sobre las codificaciones posicionales (PE), ignorando información crítica bajo ciertos escenarios. Al introducir un factor de regularización aprendible, el modelo puede compensar este sobrepeso. Otro desafío importante es el manejo de datos fuera del conjunto de entrenamiento. Investigaciones previas resolvieron este problema en entornos cerrados; el modelo propuesto intenta resolverlo utilizando Q-Learning, una técnica de aprendizaje por refuerzo en la cual el modelo recibe recompensas a medida que encuentra la mejor estrategia dentro de un conjunto de posibles acciones. En este escenario no hay restricciones para la solución posible; sin embargo, al restringir las acciones a cambios de color o de posición, ayuda a el modelo a converger hacia una solución que pueda generalizar adecuadamente más rápido.

Palabras clave: Aprendizaje por refuerzo, Inteligencia Artificial, Transformers, ARC benchmark.

Acknowledgment

First and foremost, I would like to express my deepest gratitude to Dr. Edgar Alejandro Guerrero Arroyo. His patience, guidance, and unwavering commitment to this project were truly instrumental. This journey of discovery would not have been possible without his broad and profound knowledge, which provided both direction and inspiration throughout the development of this work.

I am sincerely grateful to ITESO for equipping me with the necessary tools and fostering an academic environment that supports growth. I especially appreciate the faculty members, whose dedication and enthusiasm for teaching and learning are exceptional. Their passion made a real difference and was a constant source of motivation throughout the challenges of this research.

I would like to extend my heartfelt thanks to Riemman Ruiz and Fernando Becerra, whose unwavering support and generosity consistently went far beyond what was expected of them. Their willingness to help at every stage of this project, without hesitation or self-interest, made a significant impact on both the process and the final outcome. Their integrity, remarkable work ethic, and outstanding expertise are truly exceptional and rare to find. Working alongside individuals of such caliber has been both an honor and an inspiration.

To my colleagues Uriel Gómez, Martín Mota and Óscar Retolaza. I am grateful to have shared this journey with you. Navigating this process together made all the difference. Life gave me the opportunity to work alongside brilliant and supportive peers. From brainstorming ideas to constantly introducing me to new information relevant to my research, your input and advice made this experience not only easier, but also genuinely enjoyable. Your companionship turned challenges into learning opportunities and moments of stress into moments of growth.

To my parents, Mario and Miriam. you are the embodiment of hard work and resilience. Your love and care have always been a pillar of strength in my life. You encouraged me, time and again, to pursue my goals with determination and integrity. More than just words, you taught me through your example what it means to persevere and stay grounded. Thank you for everything — your support has shaped who I am and made this achievement possible.

To my wife, Itzel. who stood by my side every single day and night, sharing in both the struggles and the victories of this journey. Her constant presence, unwavering support, and boundless patience were essential to the completion of this work. Without her, none of this would have been possible. Thank you for being my rock, my motivation, and an unbelievable source of strength throughout it all.

In loving memory of my grandmother, Delia Bazán. Although you are no longer with us, your warmth, wisdom, and strength continue to guide me every day. You taught me the value of family, perseverance and kindness, not just through your words, but through the life you lived. Your love was a constant source of comfort and courage, and your memory has been with me every day. I dedicate this work to you, with all my love and gratitude. You are missed every day.

Agradecimientos

Antes que nada, me gustaría expresar mi más profundo agradecimiento al Dr. Edgar Alejandro Guerrero Arroyo. Su paciencia, guía y compromiso inquebrantable con este proyecto fueron verdaderamente fundamentales. Este viaje de descubrimiento no habría sido posible sin su amplio y profundo conocimiento, que proporcionó tanto dirección como inspiración a lo largo del desarrollo de este trabajo.

Estoy sinceramente agradecido con el ITESO por brindarme las herramientas necesarias y fomentar un entorno académico que apoya el crecimiento. Aprecio especialmente a los miembros del profesorado, cuya dedicación y entusiasmo por la enseñanza y el aprendizaje son excepcionales. Su pasión marcó una verdadera diferencia y fue una fuente constante de motivación durante los desafíos de esta investigación.

Quisiera extender mi más sincero agradecimiento a Riemman Ruiz y Fernando Becerra, cuyo apoyo incondicional y generosidad superaron sus obligaciones. Su disposición para ayudar en cada etapa de este proyecto, sin dudarlo, tuvo un impacto significativo tanto en el proceso como en el resultado final. Su integridad y trabajo ejemplar además del sobresaliente conocimiento son verdaderamente excepcionales y poco comunes. Trabajar junto a personas de tan alto nivel ha sido tanto un honor como una inspiración.

A mis compañeros Uriel Gómez, Martín Mota y Óscar Retolaza. Estoy muy agradecido de haber compartido este camino con ustedes. Haber atravesado este proceso juntos marcó una gran diferencia. La vida me dio la oportunidad de trabajar junto a colegas brillantes y solidarios. Desde generar ideas en conjunto hasta compartir constantemente nueva información relevante para mi investigación, su apoyo y consejos hicieron que esta experiencia fuera no solo más sencilla, sino también genuinamente enriquecedora. Su compañía convirtió los desafíos en oportunidades de aprendizaje y los momentos de estrés en momentos de crecimiento.

A mis padres, Mario y Miriam. Ustedes son el reflejo del trabajo duro y la resiliencia. Su amor y cuidado siempre han sido un pilar fundamental en mi vida. Una y otra vez me impulsaron a perseguir mis metas con determinación e integridad. Más allá de las palabras, me enseñaron con el ejemplo lo que significa perseverar y mantenerse firme. Gracias por todo. Su apoyo ha moldeado quién soy y ha hecho posible este logro.

A mi esposa, Itzel. Quien estuvo a mi lado todos los días y noches, compartiendo tanto las dificultades como las victorias de este camino. Su presencia constante, apoyo inquebrantable y paciencia infinita fueron esenciales para completar este trabajo. Sin ella, nada de esto habría sido posible. Gracias por ser mi fuerza, mi motivación y una fuente inagotable de apoyo a lo largo de todo este proceso.

En memoria de mi abuela, Delia Bazán. Aunque ya no estés con nosotros, tu calidez, sabiduría y fortaleza continúan guiándome cada día. Me enseñaste el valor de la familia, la perseverancia y la bondad, no solo con tus palabras, sino con la vida que llevaste. Tu amor fue una fuente constante de consuelo y valentía, y tu memoria me acompaña cada día. Dedico este trabajo a ti, con todo mi amor y gratitud. Te extraño todos los días.

In loving memory of my grandmother Maria Delia Bazan Carrillo, whose wisdom and kindness continue to inspire me every day.

En memoria de mi amada abuela Maria Delia Bazan Carrillo, cuya sabiduría y bondad continúan inspirándome cada día.

Contents

List of Figures

List of Tables

Notation

- $a^{(l)}$ Activation vector at layer l of the neural network.
- $z^{(l)}$ weighted sum at layer l.
- $W^{(l)}$ Weight matrix of layer l.
- $b^{(l)}$ Bias vector of layer l.
- $\sigma^{(l)}(\cdot)$ Non-linear activation function used at layer l.
- L Total number of layers in the feed-forward network.
- x Input ARC grid flattened into a sequence.
- y Ground-truth output grid for a task instance.
- $\hat{y}$ Model prediction for the output grid.
- $\mathbf{z}_i$ Embedding vector of training example i.
- N Number of training examples provided for a task.
- $\mathcal{S}$ Set of states in the Markov Decision Process (MDP).
- $\mathcal{A}(s)$ Set of actions available in state s.

- $p(s' \mid s, a)$ Transition probability from s to s' given action a.
- r_t Reward obtained at discrete time step t.
- $\gamma \in (0, 1]$ Discount factor that weights future rewards.
- $\pi(a \mid s)$ Stochastic policy: probability of selecting action a in state s.
- $Q_\theta(s, a)$ Action-value (Q-)function parameterized by θ.
- $Q_{\theta^-}(s, a)$ *Target* Q-network used for stabilization (lags behind θ).
- θ Trainable parameters of the Q-network (and, when unfrozen, of ViT-ARC).
- ϵ Exploration rate in the ϵ-greedy action-selection strategy.
- $\mathcal{M}$ Experience-replay buffer storing past transitions (s, a, r, s').
- $\mathcal{L}$ Supervised loss (cross-entropy or MSE, depending on context).
- $J(\theta)$ Reinforcement-learning objective (expected return).
- $\nabla_\theta J$ Gradient of J with respect to parameters θ.
- T Total number of optimization steps / episodes.
- k Index of a cross-validation fold or repetition.

1 Introduction

Contents

1.1 Current Challenges and Limitations of State-of-the-Art Models

State-of-the-art (SOTA) AI models (e.g. deep neural networks and large language models) have achieved impressive results in narrow domains, but they face significant challenges when aiming for artificial general intelligence (AGI). Three critical pain points are comprehension,logical reasoning, and planning. Current models often generalize poorly outside the distribution of their training data, meaning they struggle with novel problems that require reasoning or long-term strategy. books[1] As François Chollet argues, today's AI systems can "buy" high performance on specific tasks by leveraging massive training data or prior knowledge, which *masks* their true generalization capability. [2] In other words, a model might appear highly skilled at a benchmark game or dataset, but that skill does not easily transfer to new, unseen challenges a hallmark limitation on the path toward AGI.

A big limitation is the lack of abstract reasoning. Neural networks excel at pattern recognition but struggle with tasks requiring logical inference, relational reasoning, or understanding implicit rules. For example, even the most advanced large language models (LLMs) fail to solve certain reasoning puzzles that humans find trivial without extensive domain-specific prompting or training. [3] This indicates a gap in fluid intelligence the ability to quickly grasp new concepts and solve problems one hasn't seen before. Another limitation is in planning: most deep learning models do not natively plan multi-step solutions. They map inputs to outputs in a single forward pass and lack an internal model of sequential decision-making. While specialized systems like AlphaGo/AlphaZero integrate planning via Monte Carlo

[1] W. Li, Y. Xu, S. Sanner, and E. B. Khalil. Tackling the abstraction and reasoning corpus with vision transformers: the importance of 2d representation, positions, and objects. https://arxiv.org/abs/2410.06405, 2024a

[2] F. Chollet, M. Knoop, G. Kamradt, and B. Landers. Arc prize 2024: Technical report. https://arxiv.org/abs/2412.04604, 2024

[3] W. Li, Y. Xu, S. Sanner, and E. B. Khalil. Tackling the abstraction and reasoning corpus with vision transformers: the importance of 2d representation, positions, and objects. https://arxiv.org/abs/2410.06405, 2024a; and F. Chollet, M. Knoop, G. Kamradt, and B. Landers. Arc prize 2024: Technical report. https://arxiv.org/abs/2412.04604, 2024

Tree Search, [4] those solutions are narrowly tailored to games and don't generalize broadly. In open-ended environments or puzzles, current AI often cannot strategize several steps ahead without substantial human-defined scaffolding.

These challenges are clearly reflected when SOTA models are tested on benchmarks that demand reasoning and planning. A prominent example is the Abstraction and Reasoning Corpus (ARC), which is specifically designed to expose the weaknesses of current AI in generalization and reasoning. [5] Modern neural networks, including powerful vision or language transformers, have proven unable to solve the majority of ARC tasks, precisely because those tasks require understanding abstract relations from very few examples. [6] Large pretrained models do not inherently possess the necessary concepts or cognitive mechanisms to reliably crack such problems [7] given that those AI systems lack:

Generalization: They perform well on tasks they were trained on, but struggle with unseen tasks or domains, revealing limited adaptability. [8]

Reasoning: They have difficulty inferring hidden rules or relationships without exhaustive training, indicating weak *out-of-distribution* reasoning skills

Planning: They do not innately plan multi-step solutions or long-horizon strategies; any planning ability usually comes from external algorithms or massive search, not from the learned model itself.

Efficient Learning: Unlike humans, who can learn from a handful of examples or experiences, most AI models need large datasets or many trials to learn a new task. This inefficiency hampers their ability to tackle one-off challenges like those in ARC.

These limitations underscore why achieving true AGI remains elusive with current techniques. They motivate research into new architectures and approaches that could enable an AI to reason, plan, and generalize more like humans do.

1.2 *What is ARC*

The Abstraction and Reasoning Corpus (ARC) is a benchmark introduced by F. Chollet in 2019 as a test for human-like general intelligence in machines. [9] ARC consists of 1,000 unique tasks (400 training tasks and 600 evaluation tasks) that are formulated as miniature

[4] D. Silver, J. Schrittwieser, K. Simonyan, I. Antonoglou, A. Huang, A. Guez, ..., and D. Hassabis. Mastering chess and shogi by self-play with a general reinforcement learning algorithm. https://arxiv.org/abs/1712.01815, 2017

[5] W. Li, Y. Xu, S. Sanner, and E. B. Khalil. Tackling the abstraction and reasoning corpus with vision transformers: the importance of 2d representation, positions, and objects. https://arxiv.org/abs/2410.06405, 2024a

[6] W. Li, Y. Xu, S. Sanner, and E. B. Khalil. Tackling the abstraction and reasoning corpus with vision transformers: the importance of 2d representation, positions, and objects. https://arxiv.org/abs/2410.06405, 2024a

[7] A. Johnson, W. K. Vong, B. M. Lake, and T. M. Gureckis. Fast and flexible: Human program induction in abstract reasoning tasks. https://arxiv.org/abs/2103.05823, 2021

[8] F. Chollet, M. Knoop, G. Kamradt, and B. Landers. Arc prize 2024: Technical report. https://arxiv.org/abs/2412.04604, 2024

[9] F. Chollet, M. Knoop, G. Kamradt, and B. Landers. Arc prize 2024: Technical report. https://arxiv.org/abs/2412.04604, 2024

puzzles. Each task provides a few demonstration examples (typically 3 or 4) as small grid images where an input grid is transformed into a corresponding output grid. [10] The agent (human or AI) is then given a new input grid and must produce the correct output grid, using the patterns inferred from the demonstrations. These tasks are designed to require abstract reasoning, they resemble logic puzzles on IQ test. In fact, ARC's format is inspired by intelligence tests like Raven's Progressive Matrices, but with a richer variety of problem types. [11]

A distinctive aspect of ARC is that every task is unique therefore the logic or pattern varies drastically from task to task. One puzzle might require, say, counting objects and outputting a grid with that many dots, while another might involve rotating a shape or symmetrically reflecting a pattern. This diversity means an algorithm cannot simply be trained on a large subset of ARC tasks.

ARC was built so that it is impossible to prepare for the specific tasks in advance, the tasks were handcrafted to ensure a high degree of novelty and to prevent learning any statistical regularities across tasks. [12] An ARC agent must truly infer the rule on the fly for each new task by analogy with the few given examples, much like a human would.

ARC tasks are formulated to be solvable by humans with relatively little effort, [13] assuming some basic cognitive priors. The only allowed prior knowledge is the intuitive concepts that nearly all humans acquire early in life, such as: objects and their persistence, counting small numbers, basic geometry (connectivity, symmetry), simple causality, etc. [14] No task requires literacy, specific cultural knowledge, or mathematics beyond elementary arithmetic. By limiting the required background knowledge to these core concepts, ARC ensures that success in a task comes from discovering the pattern or rule in the given examples.

In Figure 1.1 we can observe Task: b8cdaf2b, e9614598, de1cd16c. All the three task have 'rules' that explain how to solve the problem, in the first task is take the columns with color pixels only in the bottom row; then from the center to the outer section left one pixel per columns for the 2nd and 4th column, with the opposite color expressed in the grid, and the move to next row and apply the same process with the 1st and 5th column. The second task add a green cross at the center of the dots, the this task is selecting the color of the section in the grid with the most columns. All of these are logical steps, however that does not mean that the logic can be learned from the task itself, in scenarios where the information needed to solve the task is not incorporated in the task itself the model won't be able to solve the problem as pretrained models are outside the scope of this thesis.

Several characteristics make ARC particularly challenging and distinctive as a benchmark. First, the state space and action space

[10] F. Chollet, M. Knoop, G. Kamradt, and B. Landers. Arc prize 2024: Technical report. https://arxiv.org/abs/2412.04604, 2024

[11] F. Chollet, M. Knoop, G. Kamradt, and B. Landers. Arc prize 2024: Technical report. https://arxiv.org/abs/2412.04604, 2024

[12] A. Dosovitskiy, L. Beyer, A. Kolesnikov, D. Weissenborn, X. Zhai, T. Unterthiner, ..., and N. Houlsby. An image is worth 16x16 words: Transformers for image recognition at scale. https://arxiv.org/abs/2010.11929, 2020

[13] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A. Rusu, Joel Veness, Marc G. Bellemare, Alex Graves, Martin Riedmiller, Andreas K. Fidjeland, Georg Ostrovski, Stig Petersen, Charles Beattie, Amir Sadik, Ioannis Antonoglou, Helen King, Dharshan Kumaran, Daan Wierstra, Shane Legg, and Demis Hassabis. Human-level control through deep reinforcement learning. *Nature*, 2015. URL https://www.nature.com/articles/nature14236

[14] A. Dosovitskiy, L. Beyer, A. Kolesnikov, D. Weissenborn, X. Zhai, T. Unterthiner, ..., and N. Houlsby. An image is worth 16x16 words: Transformers for image recognition at scale. https://arxiv.org/abs/2010.11929, 2020

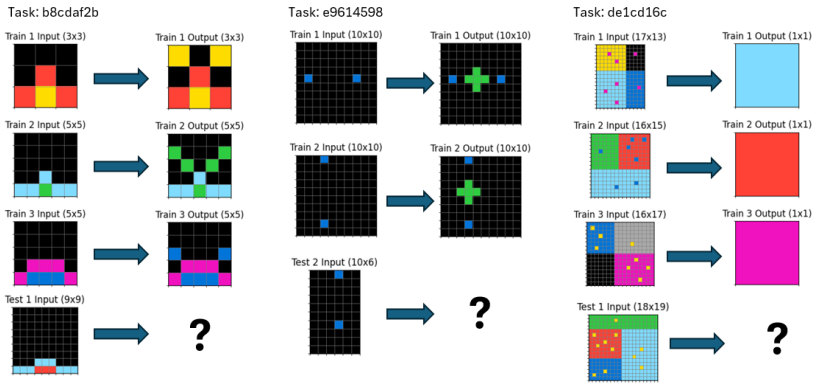


Figure 1.1: Task: b8cdaf2b - Choose the color of the section with more dots. Task: e9614598 - Fill space between blue dots with a green cross. Task: de1cd16c - Choose the color of the section with more dots.

are not explicitly given, unlike puzzles such as the Rubik's cube or chess where moves and states are well-defined, an ARC agent has to figure out a high-level representation for the problem (e.g. identify objects in the grid and their properties) and a sequence of operations to transform the input into output. This leans on the agent's ability to construct abstract representations.

Second, ARC tasks often require combining multiple steps of reasoning: an agent might have to detect a pattern and then perform a transformation conditioned on that pattern, a task could be: "Find the smallest shape and color it red, while leaving other shapes unchanged." Solving this involves at least two sub-tasks (finding smallest shape, then recoloring) a form of conditional logic that tests an AI's capacity for multi-step inference. Third, there is an emphasis on robustness: since only a few examples are given, the inferred rule must be exact. If the agent over fits to incidental features of the examples, it will likely fail on the test. Humans are adept at ignoring irrelevant details and homing in on the essential pattern (thanks to our prior knowledge and cognitive biases), but for machines this is difficult.

In an ARC task, the correct output for a new input is only achievable by finding the underlying rule (see Figure 1.1). Because each task is different, any hard-coded or learned policy tuned to one specific puzzle will fail on others. To succeed an AI must understand the problem find a function or a mix of functions that can solve it rather just mapping to a example in the training.

ARC Benchmark was released for the first time in 2019 making it 6 years now and to this point it haven't been possible to produce human level performance assessing it, making it one of the most important unsolved benchmarks in AI because it explicitly tests for generalization on novel tasks rather than performance on repetitive training scenarios. [15] Early attempts on ARC, such as a 2020 Kaggle

[15] A. Dosovitskiy, L. Beyer, A. Kolesnikov, D. Weissenborn, X. Zhai, T. Unterthiner, ..., and N. Houlsby. An image is worth 16x16 words: Transformers for image recognition at scale. https://arxiv.org/abs/2010.11929, 2020

competition, saw the best AI solvers achieve only around 20% success rate using hand-crafted symbolic programs. [16] This is drastically below human-level performance (approximately 80%), Neural approaches fail to reliably infer the correct transformations, while symbolic search faces combinatorial explosion due to the diversity of tasks.

[16] W.-D. Li, K. Hu, C. Larsen, Y. Wu, S. Alford, C. Woo, ..., and K. Ellis. Combining induction and transduction for abstract reasoning. https://arxiv.org/abs/2411.02272, 2024b

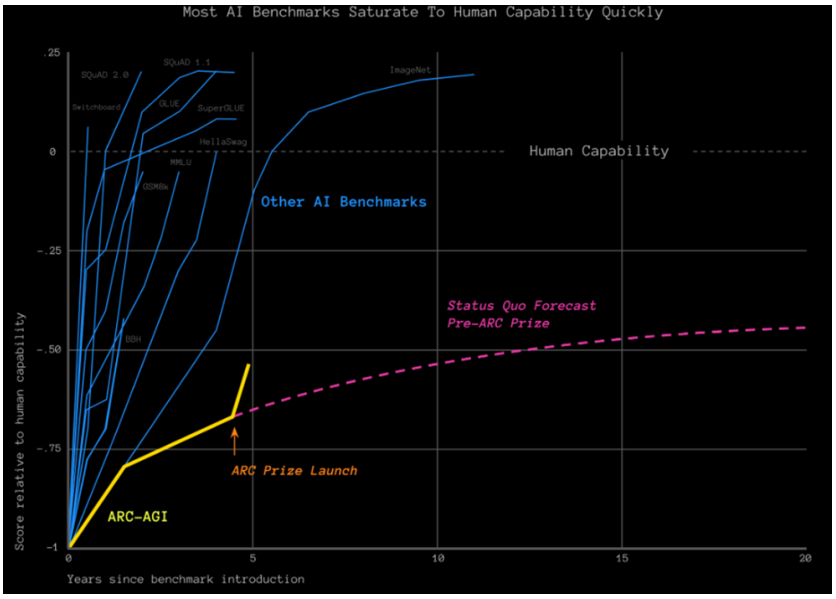


Figure 1.2: Plot show other benchmark like ImageNet, MMLU and GLUE which often in less the 5 years saturates. ARC-AGI is not showing this trend, also shows the change in progress after ARC prize launch

FIGURE 1.2 In 2024 ARC-AGI competition spurred new methods that lifted the score from about 33% to 55% on the private test set [17]. Top-performing approaches use reasoning techniques like deep learning-guided program synthesis and test-time training [18]. While these results show progress, they still fall short of the 80% target that would indicate approaching human-like proficiency.

[17] F. Chollet, M. Knoop, G. Kamradt, and B. Landers. Arc prize 2024: Technical report. https://arxiv.org/abs/2412.04604, 2024

[18] A. Johnson, W. K. Vong, B. M. Lake, and T. M. Gureckis. Fast and flexible: Human program induction in abstract reasoning tasks. https://arxiv.org/abs/2103.05823, 2021

1.3 *ViT RL as a Potential Solution*

This thesis explores traditional methodologies, identifying key limitations and refining approaches iteratively. The architectures tested included a simple multilayer perceptron and a basic ViT Transformer. Through this process, a novel Vision Transformer (ViT) combined with reinforcement learning (RL) was proposed. While it does not yet solve the Abstraction and Reasoning Corpus (ARC) benchmark, the model reveals crucial weaknesses in current methods and outlines potential research directions. The work concludes with three high-level insights that provide a solid base for future studies in visual reasoning with

transformer-based architectures.

1 - Domain-Specific Languages (DSLs). Neural networks typically operate by learning the underlying probability distribution that maps inputs to outputs, resulting in inherently probabilistic solutions. However, integrating DSLs into the modeling process has shown significant performance improvements. This is largely because DSLs constrain the solution space to a predefined set of functions, effectively shifting the task from sampling a continuous distribution to selecting or composing deterministic operations. By forcing the model to choose from a structured set of operations, DSLs can lead to more interpretable and robust solutions. [19]

[19] M. Andrews. Capturing sparks of abstraction for the arc challenge. https://arxiv.org/abs/2411.11206, 2024; and K. Singhal and G. Shroff. Conceptsearch: Towards efficient program search using llms for abstraction and reasoning corpus (arc). https://arxiv.org/abs/2412.07322, 2024

2 - Effective transmission of task-relevant information within neural network architectures. Although multilayer perceptrons (MLPs) are capable of fitting input-output mappings, their representations often lack the capacity for robust generalization, even across examples of the same Task. One primary reason is the absence of constrains that preserve domain structure. For example, flattening an input grid into a vector format erases spatial relationships. While Convolutional neural networks (CNNs) attempt to mitigate this through localized spatial filtering, they are not inherently guaranteed to retain global structural coherence, which may result in suboptimal solutions. Transformer-based architectures helps preserving critical information for finding optimal solutions leveraging on their three main components:

- Vector embeddings facilitate fine-grained, token-level representation of visual elements, preserving feature specificity at the pixel level.
- Positional encodings explicitly encode spatial configuration, maintaining information about relative and absolute positions within the grid.
- Self-attention mechanisms dynamically model dependencies and interactions between disparate regions of the input, enabling the extraction of global relational patterns.

3 - RL for expanded exploration of possible solutions. Reinforcement learning can benefit by enabling expanded exploration of the space of possible solutions. The ARC tasks require finding the underlying rule or function that maps the input to the desired output, which can be challenging given the diversity of tasks and the need for generalization. RL provides a framework for an agent to dynamically explore different strategies, [20] learn from feedback, and discover novel solutions that may not be intuitive or easily encoded in a predefined program. By coupling the information-preserving capabilities of Transformer-based architectures with the adaptive exploration of RL, the model can more effectively navigate the complex solution space and increase the

[20] W.-D. Li, K. Hu, C. Larsen, Y. Wu, S. Alford, C. Woo, ..., and K. Ellis. Combining induction and transduction for abstract reasoning. https://arxiv.org/abs/2411.02272, 2024b

likelihood of finding the correct transformation for each ARC task. This expanded search capability is crucial for overcoming the limitations of purely symbolic or neural-based approaches that may struggle to generalize across the wide range of ARC problems.

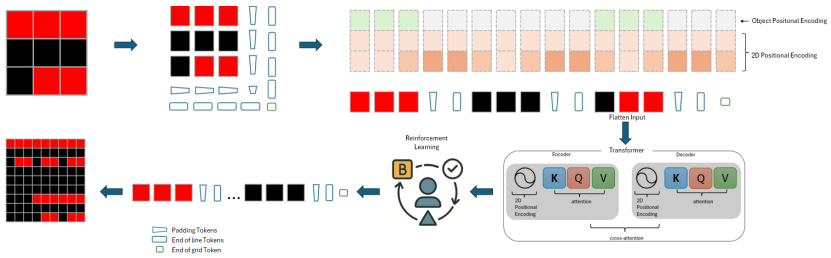


Figure 1.3: ViT-RL workflow

FIGURE 1.3 represents the overall workflow of proposed architecture. First the ARC input image is tokenized in a one pixel one input basis (flattening) and the remained input neurons are padded including end of row (EOR) and end of grid (EOG) tokens. Object positional encoding and 2D positional encoding is used to make de neural network "aware" of objects and relationship of objects in the grid space, this is space aware vector goes thought a Q-learning RL layer to explore the space of solutions after the grid can be reconstructed to the solution form.

2 Related Work

Contents

2.1 Other Approaches

To gain a deeper understanding of the challenges that ARCs represent, we tested the benchmark with various models. By highlighting the limitations of each model, we aim to present more robust alternatives. Through this process of testing and improvement, we hope to identify the best-performing model.

The significant challenges posed by the ARC benchmark, were uncovered by testing using a range of AI models, including Convolutional neural networks (CNN's), simple multilayer perceptrons (MLPs), generative adversarial networks (GANs), and standard visual transformers (ViTs). The evaluation progressively advanced from simpler to more complex models. Each model was carefully assessed to identify its inherent limitations, particularly in terms of generalization, abstract reasoning, and multi-step planning capabilities.

The analysis revealed specific weaknesses within each architecture: CNN's excelled in pattern recognition but struggled significantly due to their patch-based approach, which could isolate structures within the image, limiting their ability to generalize meaningful information to new tasks. MLPs lacked the capacity for transferring spatial structures effectively. GANs performed inconsistently, mainly due to difficulties in generating precise relational inferences. Standard ViTs demonstrated

better overall performance but were inherently limited by their heavy reliance on vector embeddings.

Building upon these insights, a novel, enhanced ViT architecture incorporating positional encoding regularization and reinforcement learning (ViT-RL) was proposed. This approach explicitly addresses the deficiencies identified in the other models, particularly the gaps in reasoning, planning, and efficient generalization from limited examples. Through iterative improvements and rigorous testing against the ARC benchmark, the aim is to develop a more robust, versatile, and effective model capable of achieving higher performance.

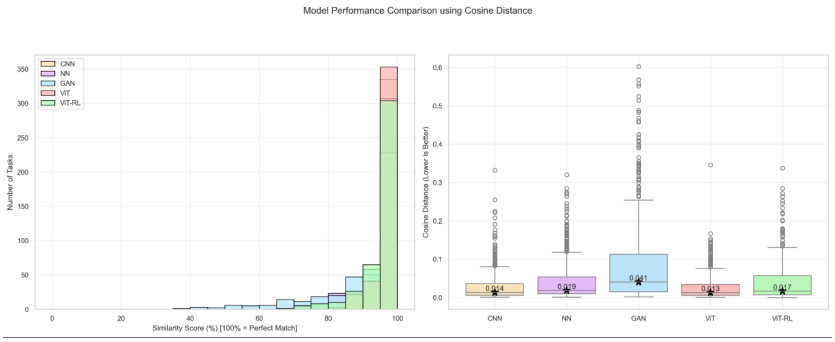


Figure 2.1: Model performance comparative for CNN, NN, GAN. ViT, ViT-RL

FIGURE 2.1 shows model performance comparative: list the five model used for comparative; a convolutional neural network, a simple multilayer perceptron, a generative adversarial network, a visual transformer and the proposed model a visual transformer with positional encoding regularization and reinforcement learning. Left table (average error per task) The horizontal axis show the average error score based on cosine similarity distance. The vertical axis show the number of task evaluated. Right table (error distribution per model). The start show the average error based on cosine similarity. CNN, ViT and ViT-RL are the best model based on average error and error sparsity.

2.2 *Multilayer perceptrons (MLPs)*

An impressive property of a fully connected multilayer perceptron is its ability to represent any solution for a given mapping problem $x \rightarrow y$. [1] Therefore, the solution for a task could, in theory, be stored in an MLP. However, a key limitation is that a simple input and output layer with a fixed number of neurons cannot capture much of the rich information embedded in the grid.

Some of the critical information lost in MLPs includes the size of the

[1] George Cybenko. Approximation by superpositions of a sigmoidal function. *Mathematics of Control, Signals and Systems*, 1989

grids, underlying structures within the grids, and relationships between pixels. Essentially, the only information that is directly passed through in MLPs is the color of each pixel; all other aspects are assumed to be fixed or irrelevant. This limitation severely affects the model's ability to generalize or understand the spatial and relational context inherent in many visual tasks.

Let $a^{(0)}$ denote the input vector. For each layer $l \in \{1, \ldots, L\}$, the following operations are performed:

$$\begin{aligned}
\text{Input layer:} \quad & a^{(0)} = x, && (2.1)\\
\text{Deep layers:} \quad & z^{(l)} = W^{(l)} a^{(l-1)} + b^{(l)}, && (2.2)\\
& a^{(l)} = \sigma^{(l)}(z^{(l)}), \quad l = 1, 2, \ldots, L-1, && (2.3)\\
\text{Output layer:} \quad & z^{(L)} = W^{(L)} a^{(L-1)} + b^{(L)}, && (2.4)\\
& \hat{y} = a^{(L)} = \sigma^{(L)}(z^{(L)}), && (2.5)
\end{aligned}$$

where:

- x is the input vector.
- $a^{(l)}$ is the activation of layer l.
- $z^{(l)}$ is the pre-activation (linear output) of layer l.
- $W^{(l)}$ and $b^{(l)}$ are the weights and biases of layer l.
- $\sigma^{(l)}$ is the activation function (e.g., ReLU, sigmoid) at layer l.
- $\hat{y}$ is the output of the network.
- L is the total number of layers including the output layer.

2.3 *Convolutional Neural Network*

A Convolutional neural network (CNN) [2] uses filters to extract spatial information from an image (grid), addressing one of the major limitations of multilayer perceptrons (MLPs). Unlike MLPs, CNN's can process both structural and pixel color information, allowing for richer representations.

However, these filters are predefined and fixed, meaning they may not generalize well to unseen tasks. In some cases, filters may segment the grid in ways that disrupt the logical structure needed to solve a problem. While CNN's inherit the mapping capabilities of MLPs and can often solve tasks more efficiently due to the added structural information they still face key limitations. For example, CNN's might miss implicit cues in pixel colors, lose important structural information

[2] Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 1998

by cutting through shapes, or fail to capture relational patterns between different structures.

Thus, despite offering improvements over MLPs, CNN's remain constrained in their ability to generalize and reason about complex, unseen tasks.

Let $a^{(0)}$ denote the input image or feature map. For each layer $l \in \{1, \ldots, L\}$, the following operations are performed:

$$\text{Input layer:} \quad a^{(0)} = x, \tag{2.6}$$

$$\text{Convolutional layers:} \quad z^{(l)} = W^{(l)} * a^{(l-1)} + b^{(l)}, \tag{2.7}$$

$$a^{(l)} = \sigma^{(l)}(z^{(l)}), \quad l = 1, 2, \ldots, L-1, \tag{2.8}$$

$$\text{Output layer:} \quad z^{(L)} = W^{(L)} * a^{(L-1)} + b^{(L)}, \tag{2.9}$$

$$\hat{y} = a^{(L)} = \sigma^{(L)}(z^{(L)}), \tag{2.10}$$

where:

- x is the input grid.
- $a^{(l)}$ is the activation (feature map) at layer l.
- $z^{(l)}$ is the pre-activation result of the convolution at layer l.
- $W^{(l)}$ and $b^{(l)}$ are the Convolutional filters and biases at layer l.
- $*$ denotes the convolution operation.
- $\sigma^{(l)}$ is the activation function applied element wise.
- $\hat{y}$ is the final output.
- L is the total number of layers including the output layer.

2.4 *Generative Adversarial Network*

Generative adversarial networks (GANs) [3] address another major challenge of the ARC benchmark: solving tasks based on limited examples. A central argument of the benchmark is that all the necessary information to solve a task is contained within just a few provided examples. However, with such a small sample size, it is possible that a task might have multiple valid solutions, or worse, that a model finds a solution that works for the given test cases but fails to generalize to new, unseen instances.

GANs attempt to mitigate this issue by generating new examples that resemble the original task, aiming to identify a solution capable of solving both the seen and unseen cases. Despite this advantage,

[3] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial nets. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2014

GANs still face many of the same limitations as MLPs and CNN's. They often struggle to capture and transfer essential information such as pixel color, grid shape, internal structures, and implicit relationships. Without incorporating these elements effectively, the new examples generated by GANs may rely on incorrect assumptions. As a result, the model may perform well on familiar tasks but fail to generalize to different or more complex scenarios.

Let z denote a random noise vector sampled from a prior distribution. A GAN consists of two neural networks: a generator G and a discriminator D.

$$\text{Generator:} \quad a_G^{(0)} = z, \tag{2.11}$$

$$z_G^{(l)} = W_G^{(l)} a_G^{(l-1)} + b_G^{(l)}, \tag{2.12}$$

$$a_G^{(l)} = \sigma^{(l)}(z_G^{(l)}), \quad l = 1, 2, \ldots, L_G, \tag{2.13}$$

$$\hat{x} = a_G^{(L_G)} = G(z), \tag{2.14}$$

$$\text{Discriminator:} \quad a_D^{(0)} = x \quad \text{or} \quad \hat{x}, \tag{2.15}$$

$$z_D^{(l)} = W_D^{(l)} a_D^{(l-1)} + b_D^{(l)}, \tag{2.16}$$

$$a_D^{(l)} = \sigma^{(l)}(z_D^{(l)}), \quad l = 1, 2, \ldots, L_D, \tag{2.17}$$

$$\hat{y} = a_D^{(L_D)} = D(x), \tag{2.18}$$

where:

- z is the input noise vector.
- $G(z)$ is the generated (fake) data sample.
- x is a real data sample from the dataset.
- $a_G^{(l)}$, $z_G^{(l)}$, $W_G^{(l)}$, and $b_G^{(l)}$ are the activations, pre-activations, weights, and biases of the generator.
- $a_D^{(l)}$, $z_D^{(l)}$, $W_D^{(l)}$, and $b_D^{(l)}$ are the corresponding terms for the discriminator.
- $\hat{y}$ is the discriminator's output probability (real vs. fake).
- L_G and L_D are the number of layers in the generator and discriminator, respectively.

2.4.1 *Visual Transformers*

For Visual Transformers (ViT) are different from traditional transformers. In text transformers, input text is divided into chunks called "tokens." These tokens are processed through three main components: vector embeddings, positional encoding, and attention mechanisms. Visual transformers apply a similar approach to images, breaking them into patches that function similarly to text tokens. These patches are then passed through the same processing steps used in text transformers: vector embeddings, positional encoding, and attention mechanisms. [4]

[4] A. Dosovitskiy, L. Beyer, A. Kolesnikov, D. Weissenborn, X. Zhai, T. Unterthiner, ..., and N. Houlsby. An image is worth 16x16 words: Transformers for image recognition at scale. https://arxiv.org/abs/2010.11929, 2020

Experimental results show that visual transformers address many of the limitations seen in MLPs and CNN's, especially in terms of information representation. Pixel colors are converted into semantic vectors through vector embeddings, allowing the model to capture how colors interact with each other and with structures in the grid. Positional encoding informs the network of spatial relationships, independent of color. Finally, the attention mechanism merges color and positional information to uncover relationships between elements, ensuring no relevant task information is lost.

Despite these advances, visual transformers still face challenges. Having access to all relevant information does not guarantee it will be managed effectively or used appropriately.

Research by Wenhao Li et al. found that visual transformers tend to prioritize color information over positional cues. [5] ARC tasks are diverse, and in roughly half of them, positional information is more critical than color. While traditional visual transformers perform well on color-dominant tasks, they underperformed when positional understanding is key. The Positional Encoding Mixer (PE Mixer) is used when problems involve a mix of color and positional information. Dynamically adjusts the balance between color and position through two learnable parameters that weight the relevance of either color or position, thereby improving overall model performance on the ARC benchmark.

[5] W. Li, Y. Xu, S. Sanner, and E. B. Khalil. Tackling the abstraction and reasoning corpus with vision transformers: the importance of 2d representation, positions, and objects. https://arxiv.org/abs/2410.06405, 2024a

Even with well-represented and balanced information, reasoning remains a significant obstacle. This limitation was partially addressed by GANs, which attempted to model complex patterns. However, to better handle the structured information provided by the transformer, reinforcement learning is introduced as a more effective alternative to GANs for reasoning and decision-making.

Let $x \in \mathbb{R}^{H \times W \times C}$ be an input image, where H, W and C are the height, width and number of channels, respectively. The Vision Transformer processes the image in three stages:

1. Patch + positional embedding

$$\text{Extract } N = \frac{HW}{P^2} \text{ non-overlapping patches of size } P \times P: \; x \longrightarrow x_{\text{patch}} \in \mathbb{R}^{N \times P^2 C}, \tag{2.19}$$

$$\text{Linear projection to the hidden size } D: \quad x_p = x_{\text{patch}}\, E,\; E \in \mathbb{R}^{P^2 C \times D},\; x_p \in \mathbb{R}^{N \times D}, \tag{2.20}$$

$$\text{Prepend a learnable class token } x_{\text{cls}} \in \mathbb{R}^{1 \times D}, \text{ add positions:} \tag{2.21}$$

$$\boxed{z^{(0)} = [\, x_{\text{cls}};\, x_p^1; \ldots; x_p^N \,] + E_{\text{pos}}}, \qquad E_{\text{pos}} \in \mathbb{R}^{(N+1) \times D}. \tag{2.22}$$

2. Transformer encoder ($l = 1, \ldots, L$)

$$z' = \text{MSA}\big(\text{LN}(z^{(l-1)})\big) + z^{(l-1)}, \tag{2.23}$$

$$z^{(l)} = \text{MLP}\big(\text{LN}(z')\big) + z'. \tag{2.24}$$

3. Classification head

$$\hat{y} = \text{MLP}_{\text{head}}\big(z_0^{(L)}\big), \tag{2.25}$$

where:

- P patch side length; $N = \frac{HW}{P^2}$ – number of patches.
- D hidden dimension after the linear projection.
- x_p projected patch embeddings.
- E_{pos} – learnable positional encodings (one extra row for the class token).
- MSA – multi-head self-attention; LN – layer normalization; FFN – position-wise feed-forward network (uses GELU and dropout in practice).
- $\hat{y}$ output prediction (logits or class probabilities).
- $z_0^{(L)}$ denotes the embedding of the class token (index 0) after the final encoder block.

2.4.2 *Model Architecture and Training Settings*

This table summarizes the architectures used in this thesis, highlighting the total number of parameters and the specific type of model. The ViT and ViT-RL architectures both share the same parameter count but differ in their learning approaches, with the latter incorporating reinforcement learning strategies.

Model	Total Parameters	Architecture Type
ViT-RL	6,826,240	Vision Transformer with Reinforcement Learning
ViT	6,826,240	Vision Transformer
Neural Network	923,012	Fully Connected Neural Network
GAN	1,727,362	Generative Adversarial Network
CNN	6,486,656	Convolutional Neural Network

Table 2.1: Model architecture and parameter details.

Model	Epochs	Batch	LR	Optimizer
ViT-RL	1000	16	0.001	AdamW
ViT	100	16	0.001	AdamW
Neural Net	100	32	0.001	Adam
GAN	200	64	0.0002	Adam
CNN	100	32	0.001	Adam

Table 2.2: Training configuration for each model.

2.4.3 *Evaluation Metrics Comparison*

This table presents a comparison of pixel accuracy (Pxl Acc.), matrix accuracy (Mtx Acc.), mean squared error (MSE), and maximum error (Max Err.) provide a comprehensive view of model effectiveness. Notably, ViT-RL achieves the highest pixel accuracy with the lowest error percentage.

Model	Pxl Acc.	Mtx Acc.	MSE	Max Err.	Size	Err. %
CNN	0.6437	0.00	5.5318	12.0000	419	35.63%
NN	0.0162	0.00	21.7957	10.3372	419	98.38%
GAN	0.4163	0.00	15.7997	12.0000	419	58.37%
ViT	0.0216	0.00	19.9959	9.3198	419	97.84%
ViT-RL	0.8690	0.00	nan	nan	419	13.10%

Table 2.3: Accuracy and error metrics for each model. Values marked as `nan` (not a number) denote entries that are not applicable for the corresponding model.

Model	Corr. Pred.	Mean Err.	Pxl Prec.	Pxl Recall	Pxl F1
CNN	0	nan	nan	nan	nan
NN	0	4.4085	0.0758	0.0939	0.0296
GAN	0	2.0621	nan	nan	nan
ViT	0	4.1494	0.0354	0.1315	0.0501
ViT-RL	0	nan	0.2683	0.2963	0.2808

Table 2.4: Prediction metrics and pixel-wise statistics. Values marked as nan (not a number) denote entries that are not applicable for the corresponding model.

3 ViT-RL

Contents

The presented algorithm is a modified version of a simple Visual Transformer (ViT) that integrates spatial information using a 2D Relative Positional Bias module, combined with a "PE mixer" strategy that fuses pixel information with positional embeddings. The model is trained by minimizing the mean squared error (MSE) between the predicted Q-values and a one-hot target encoding. Evaluation metrics range from pixel accuracy to cosine similarity between predictions and targets.

The input vector embedding and positional encoding component transform grid inputs into a latent space using a linear layer and GELU activation. This process is enhanced by learned absolute positional embeddings and a relative positional bias that captures spatial relationships. The 2D Relative Positional Bias module generates biases based on differences in row and column indices, which augments the attention mechanism by providing a sense of relative positioning.

During pre processing, grid sizes are standardized by calculating the maximum dimensions across samples and adding appropriate border tokens as padding.

The training loop minimizes the MSE loss on the Q-values using the AdamW optimizer, supported by a cosine annealing learning rate scheduler to improve convergence. Evaluation uses a combination of classification and regression metrics to assess model performance.

The model consists of 23.9 million parameters, eight attention heads, a hidden dimension of 256, and a batch size of 8.

3.0.1 Dimension Calculation

The size of the grid varies between tasks and represents an additional learnable component for the ViT. In experiments where transformers predicted the End-Of-Grid (*EOG*) token, it was common for them to fail in estimating the correct grid size. To address this challenge, a fixed maximum grid size is used, determined by identifying the largest dimensions across all samples.

To determine this maximum grid size, an exhaustive exploration of grid sizes for all examples and solutions across both the training and testing datasets was performed. This ensured that all examples could fit within the maximum grid size and therefore could be predicted correctly. As a result of this analysis, we determined the maximum grid size to be 30×30, which was chosen to ensure complete coverage of all samples.

3.0.2 Padding Strategy

All neural networks consist of an input layer, one or more hidden layers, and an output layer. In most architectures, the input and output layers have fixed sizes (i.e., a fixed number of neurons). This constraint has important implications for model behavior. For example, when layer sizes are fixed, the structure of the data cannot change. In the famous Iris exercise, there are two neurons in the input layer (petal width, sepal width) and three neurons in the output layer (Setosa, Versicolor, and Virginica) [1]. Consequently, the data fed to the model must have exactly two features, and there can only be three possible outputs. If additional features (inputs) or classes (outputs) become available, the model cannot process them. Even when the feature set and output classes remain unchanged, another requirement persists: features must always be presented in the same order. If the model was trained with petal width first and sepal width second, then at inference time petal width must still be first and sepal width second. The output layer imposes a similar ordering constraint: Setosa, Versicolor, and Virginica will always be output in that sequence.

[1] UCI Machine Learning Repository. Iris dataset example. https://archive.ics.uci.edu/ml/datasets/iris, 1988

These fixed-size properties pose a challenge when the data structure itself carries essential information. For instance, suppose petal and sepal widths are measured from a rotated image, such that sepal width might sometimes precede petal width. The model must be aware of the semantic content regardless of which input neuron receives a given feature.

Recurrent neural networks (RNNs), long short-term memory networks (LSTMs), and related sequence models remove the fixed-size restriction on inputs and outputs by reusing the same parameters across time steps and maintaining an internal hidden state that carries

contextual information forward. This recurrent mechanism allows them to process sequences of arbitrary length while inherently exploiting the temporal order of inputs. However, because each time step depends on the previous hidden state, these architectures can be sensitive to the ordering of data and remain order-aware rather than order-agnostic. Additionally, the recurrent connections introduce challenges in training: gradients propagated through many time steps can vanish or explode, leading to neuron saturation or unstable learning dynamics.

Through experiments with variable-length models (RNNs and LSTMs), we observed that the requirement to learn sequence lengths added an extra burden, reducing the model's predictive performance on grid transformations.

In this work, to handle grid-based tasks, we propose a fixed-size, one-dimensional padding method. For each task, we determine the maximum input and output sequence lengths across all examples; shorter sequences are padded to match this maximum length.

Our grids consist of ten colors, represented by the integers 0 through 9. We introduce three special tokens: 10 for padding, 11 for end of row (*EOR*), and 12 for end of grid (*EOG*).

Flattening a $2D$ grid into a $1D$ vector involves mapping each cell at row i and column j (zero-based indices) to a unique position k in the sequence using a row-major ordering:

$$k = i \times W + j,$$

where W is the grid width (number of columns). For example, in a 4×4 grid, the value in the cell position $(2, 1)$ becomes position 9 in k flattened

$$k = 2 \times 4 + 1 = 9.$$

In the flattened vector. This transformation converts the spatial layout into a linear sequence but severs the explicit boundary between rows.

Without demarcation, the model cannot tell where one row ends and the next begins. To address this, we insert an *EOR* token (11) immediately after each row's W color tokens, and an *EOG* token (12) at the end of the final row.

During embedding, each token $(0\text{–}9, 10\text{–}12)$ is mapped to a learned vector. The positions of *EOR* and *EOG* explicitly mark row boundaries, allowing the model to infer row indices by counting the number of tokens between markers, and column indices by the token's offset within each segment. Padding tokens (10) are added after *EOG* or within rows when necessary to reach a uniform maximum length

across examples. In this way, we encode two-dimensional positional information within a one-dimensional token sequence, enabling the model to reconstruct the original grid structure and reason about spatial relationships.

This approach addresses the fixed-size constraint on input and output layers: we can employ a fixed-size architecture while encoding structural information within the token sequence.

However, as in the Iris example, this issue is a form of spatial translation variance: the model must apply identical logic regardless of where a pattern appears. In literature, this challenge is often referred to as achieving translation invariance (or translation equivariance when intermediate representations transform in predictable ways). For example, if a blue square appears at coordinates $[(2,7),(2,8),(3,7),(3,8)]$ versus $[(2,2),(2,3),(3,2),(3,3)]$, the network should perform the same transformation.

Fully connected (MLP) architectures lack built-in translation invariance because each input neuron has a distinct weight vector. Convolutional neural networks (CNN's) overcome this by sharing filter weights across spatial locations: a learned kernel slides over the grid, detecting features invariant. Group-equivariant CNN's extend this concept to other transformations (e.g., rotations and reflections). Attention-based models, such as Vision Transformers, can also handle positional variability by incorporating relative positional encodings, which explicitly encode the offset between query and key positions rather than absolute coordinates.

These methods ensure that learned operations apply uniformly across locations, effectively solving the order-of-information problem.

FIGURE 3.1 image show two input $-$ output grid that are semantically the same but the location of the object subject to color change is located in a different section of the spatial grid

While the implementation of the padding to identify when to add the EOR token we use the fallowing process:

$$P(i,j) \text{ s.t. } \begin{cases} EOR & \text{if } i = 0 \text{ or } i = H_{\max} - 1 \text{ or } j = 0 \text{ or } j = W_{\max} - 1. \\ G(i-1, j-1) & \text{otherwise,} \end{cases}$$

where:

- EOR is used to mark the borders of the rows.
- $P(i,j)$ represents the padded grid, which has dimensions $H_{\max} \times W_{\max}$, where $H_{\max}$ and $W_{\max}$ are the maximum height and width of any sample.

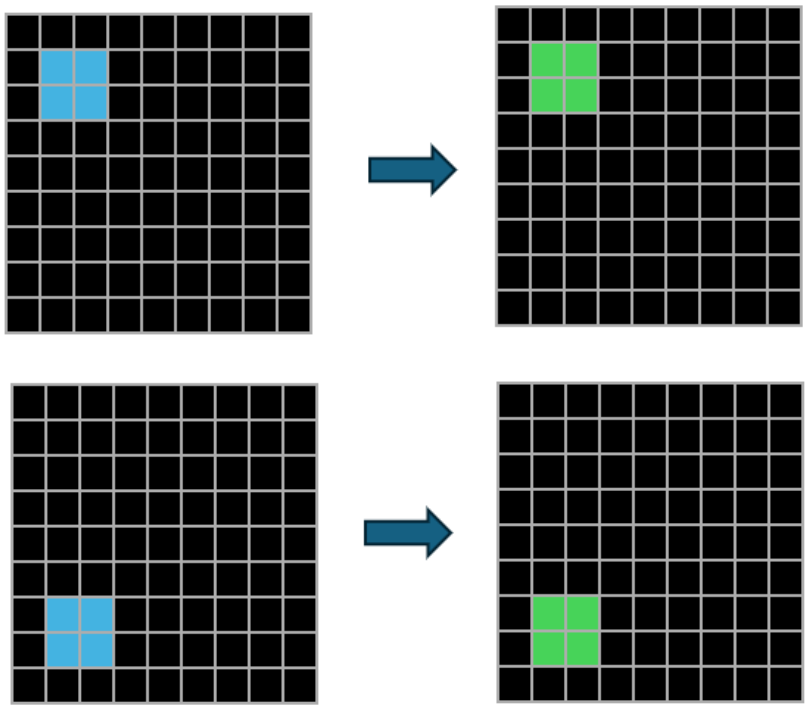


Figure 3.1: Padded Grid

- $G(i-1, j-1)$ represents the original grid (without padding), indexed from $(0,0)$ up to $(H_{\max}-2, W_{\max}-2)$.

The padding process effectively surrounds the original grid with a one-cell border filled with the *EOR* token.

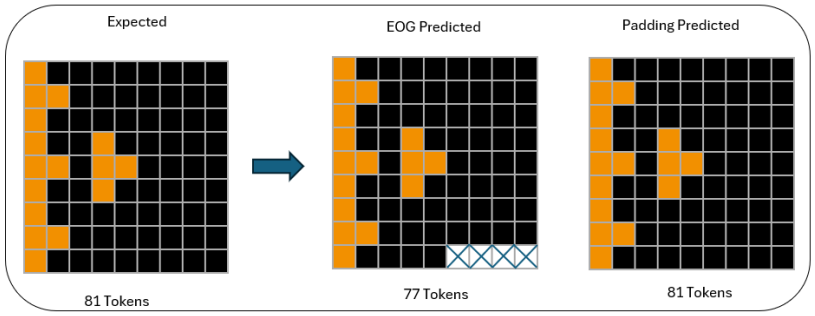


Figure 3.2: Grid padding

FIGURE 3.2 we can see on the left the correct shape of the grid for the solution on the left two examples; *EOG* Predicted where it fails to predict the right shape of the grid and the Padding Predicted where the grid shape is fixed by the padding.

Another critical issue known in the literature as numerical encoding bias or feature-scale distortion arises when continuous or ordinal-valued features impose an arbitrary metric geometry into a model's input space. Because neural networks treat inputs as points in $\mathbb{R}^n$, the numerical distances between feature values directly influence gradient updates and similarity computations, potentially conflating semantic differences with simple numeric magnitude. For example, encoding two houses with numbers 5 and 15 implies a ten-unit distance in feature space, even if the real-world separation is three houses apart. This distortion can degrade performance, particularly in tasks sensitive to relative differences rather than absolute values.

To mitigate numerical encoding bias, practitioners first apply feature scaling methods such as min–max normalization or z-score standardization, ensuring that all input features occupy comparable numerical ranges and thus preventing any single feature from disproportionately influencing gradient updates. In cases involving discrete or continuous variables with complex semantic relationships, embedding layers may be introduced. These embeddings learn vector representations that reflect feature semantics rather than raw magnitudes, decoupling the model's metric assumptions from the original numerical scale. Metric learning approaches further refine this geometry by using contrastive or triplet loss objectives that encourage semantically similar examples to lie closer in the embedding space, irrespective of their original numeric values. Additionally, regularization techniques such as weight decay, dropout, and batch or layer normalization promote smoother weight distributions and help mitigate sensitivity to input scale.

3.0.3 *Embedding*

Embeddings are a fundamental component in many neural network architectures that operate on discrete input symbols such as words, pixels, or tokens. An embedding is a learnable mapping from a finite set of discrete items into a continuous vector space. This means that each symbolic input is represented as a dense vector in $\mathbb{R}^D$, where D is the embedding size. The goal is for these vectors to capture semantic or structural information about the inputs.

For example, in natural language processing, embeddings can capture syntactic roles or semantic similarity between words.

In the context of ARC, embeddings allow us to inject semantic information into each token or pixel. Each color in an ARC task can correspond to a specific logic or transformation. For instance, blue might represent an "add" operation, while red could represent

"subtract". Though these associations are not hard-coded, they can be learned implicitly by the model during training through backpropagation. The embedding layer thus gives the model the flexibility to associate meaningful vector representations to such symbols.

From the previous padding strategy, each symbol is mapped to one of 13 possible discrete tokens: 0 to 9 for color/pixel values, and 10, 11, 12 for special tokens (padding, End-of-Row, End-of-Grid). These are integer-coded inputs, so the resulting input tensor has shape (B, S), where B is the batch size and S is the sequence length. Each element $x[b, s]$ is an integer in the range $[0, 12]$.

The model uses a learnable embedding matrix:

$$W_{\text{emb}} \in \mathbb{R}^{13 \times D}, \quad D = \text{hidden\_size},$$

where each of the 13 rows corresponds to a unique token class, and each row is a vector in $\mathbb{R}^D$, the embedding space.

For each token index $t_{b,s} \in \{0, \ldots, 12\}$, the embedding layer returns:

$$\text{emb}_{b,s,:} = W_{\text{emb}}[t_{b,s}] \in \mathbb{R}^D,$$

and thus the full embedding tensor is:

$$\text{emb} \in \mathbb{R}^{B \times S \times D},$$

These vectors are initialized randomly and refined during training. This step transforms symbolic, non-differentiable inputs into continuous representations, allowing the model to learn using gradient-based optimization.

3.0.4 *Absolute 2D Sinusoidal PE*

As the problem deals with a visual representation, it is critical to inform the model with spatial awareness. Transformer models are permutation-invariant, meaning they do not inherently understand the order or spatial position of input elements unless this information is explicitly provided. This is achieved through positional encoding.

The canonical 1D sinusoidal positional encoding, introduced by Vaswani et al. [2], maps discrete token positions in a sequence of length S to continuous representations that can be integrated with learned embeddings. This encoding is well-suited for models that process language or similarly structured sequences, where absolute positional information is important.

[2] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin. Attention is all you need. https://arxiv.org/abs/1706.03762, 2017

The core mechanism begins with the construction of an inverse frequency spectrum:

$$\text{inv\_freq}_k = \frac{1}{10000^{2k/D}}, \quad k = 0, 1, \ldots, \frac{D}{2} - 1,$$

Here, D denotes the total embedding dimensionality, and k indexes the dimensions of the encoding vector. This design ensures that each dimension is sensitive to a distinct frequency band: lower k yields higher-frequency (finer granularity) encodings, while higher k corresponds to lower-frequency (coarser granularity) encodings.This hierarchical encoding facilitates multi-scale spatial reasoning, enabling the model to simultaneously track fine local variation and broader positional trends.

For a given position p, the positional embedding vector is constructed using alternating sine and cosine functions:

$$\text{PE}(p)_{2k} = \sin(p \cdot \text{inv\_freq}_k), \quad \text{PE}(p)_{2k+1} = \cos(p \cdot \text{inv\_freq}_k),$$

This results in a positional encoding matrix of shape (S, D), which is added to the learned embeddings:

$$h_p = \text{Emb}(x_p) + \text{PE}(p),$$

Although this positional encoding scheme provides a deterministic and parameter-free mechanism for injecting order into the sequence, it exhibits limitations in tasks where relative positioning and relational inductive bias are pivotal. Specifically, it lacks explicit modeling of relative distances between tokens and entangles positional and semantic content in an inseparable manner. These limitations become especially important in domains like the ARC Task, where reasoning about spatial and structural transformations requires disentangling positional attributes from content representations and explicitly attending to relative configuration.

Consequently, recent approaches have explored alternatives to absolute sinusoidal encodings, including relative, learned, or hybrid strategies that offer more expressive and task-aligned representations.

2D Positional Encoding for Grids

To address spatial reasoning in 2D, we modify the encoding scheme to reflect the structure of 2D grids. From the padding strategy, the input is treated as a 2D grid of height h and width w. Flattening is performed row-wise, converting the grid into a 1D sequence suitable for transformer models.

To handle the 2D structure, we split the embedding vector into two halves:

- The **first half** (size $D/2$) encodes the **row position** i — vertical spatial information.
- The **second half** (also size $D/2$) encodes the **column position** j — horizontal spatial information.

This separation allows the model to treat the two spatial axes independently, which is crucial because the position in an image is inherently two-dimensional. The interaction between vertical and horizontal positioning often plays a key role in tasks such as object localization and spatial reasoning.

Using sinusoidal functions on each axis provides smooth variation with position while maintaining uniqueness and scale-invariance. For example, $\sin(x + y)$ supports representing relative offsets.

Because row and column encodings are generated from different frequency bases and placed in disjoint halves of the vector, this also makes their representations orthogonal and disentangled meaning the model can distinguish and manipulate them independently. This is especially useful in for object localization: understanding "where" in an image something appears and relational reasoning: knowing not just content, but where it sits relative to other content like in the ARC tasks.

Row Encoding for the i-th row:

$$\text{inv\_freq}_k = \frac{1}{10000^{2k/(D/2)}}, \quad k = 0, 1, \ldots, \frac{D}{2} - 1,$$

$$\text{PE}_{\text{row}}(i)_{2k} = \sin(i \cdot \text{inv\_freq}_k), \quad \text{PE}_{\text{row}}(i)_{2k+1} = \cos(i \cdot \text{inv\_freq}_k).$$

Column Encoding for the j-th column:

$$\text{inv\_freq}_k = \frac{1}{10000^{2k/(D/2)}}, \quad k = 0, 1, \ldots, \frac{D}{2} - 1,$$

$$\text{PE}_{\text{col}}(j)_{2k} = \sin(j \cdot \text{inv\_freq}_k), \quad \text{PE}_{\text{col}}(j)_{2k+1} = \cos(j \cdot \text{inv\_freq}_k).$$

Final 2D Positional Encoding:

$$\text{PE}[i, j] = [\text{PE}_{\text{row}}(i) \,\|\, \text{PE}_{\text{col}}(j)] \in \mathbb{R}^D,$$

Here, $\|$ denotes vector concatenation. This 2D positional encoding supports both absolute spatial location and relative configuration awareness, enabling effective processing of grid-based visual reasoning tasks such as those found in ARC.

3.0.5 *Positional Encoding Mixer (PE Mixer)*

In the ARC task, the importance of pixel content versus spatial position can vary significantly across different problems. In some cases, the specific color or value of a pixel is the most relevant feature, while in others, the spatial location of that pixel carries critical information. To accommodate this variability, the model employs a *weighted combination* of input embeddings and positional embeddings. This mechanism enables the model to dynamically adapt to the specific demands of each task.

The final token representation is computed as:

$$x' = \alpha \cdot E_{\text{in}}(x) + \beta \cdot E_{\text{pos}},$$

where:

- $E_{\text{in}}(x)$ is the embedding of the input pixel value,
- E_{pos} is the absolute positional embedding,
- α and β are learnable vectors of the same dimensionality as the embeddings.

The multiplications are performed element-wise. This formulation allows the model to control the relative influence of content versus position: if α dominates, the model emphasizes pixel identity; if β is stronger, the model prioritizes spatial layout. This modulation is critical in ARC, where some tasks depend more on structure, while others focus on symbolic content.

An additional advantage arises from the structure of the positional encoding itself. In the 2D setting, the positional vector is typically formed by concatenating a row encoding and a column encoding:

$$E_{\text{pos}} = [\text{PE}_{\text{row}}(i) \,\|\, \text{PE}_{\text{col}}(j)]\,,$$

This separation maintains the distinction between vertical and horizontal spatial directions. As a result, the model gains the ability to emphasize or suppress specific spatial axes independently. This improves its reasoning capabilities when working with structured grid layouts.

Compared to approaches that simply add or concatenate embeddings without modulation, this strategy offers greater flexibility. By learning the relative weighting between semantic content and positional structure, the model can better adapt to the inductive biases present in different ARC tasks.

3.0.6 *Attention mechanism and 2D Relative Positional Bias*

The attention mechanism allows neural networks to dynamically focus on specific parts of an input sequence and understand relationships between different elements—including relationships between a token and itself. Rather than processing all inputs uniformly, attention enables the model to weigh the importance of each element based on context.

There are several types of attention mechanisms, such as additive attention, multiplicative (dot-product) attention, self-attention, and cross-attention each differing in how the relevance scores are computed among them. For this model the Query-Key-Value (QKV) formulation is implemented, a common formulation for Transformer architectures as has proven especially powerful.

In QKV attention, each input element from the vector embedding after went through the Absolute 2D Positional Encoding (2D APE) x' is projected into three distinct vectors: a query (Q), a key (K), and a value (V). The interaction between queries and keys determines attention weights, which are computed based on a similarity index by using a scaled dot product. This index measures how closely aligned a query is with each key, capturing their semantic or spatial relevance. The resulting attention weights quantify the importance of each key with respect to the query and are subsequently used to combine the corresponding value vectors in a weighted sum. This similarity-driven mechanism ensures that each output representation selectively integrates information from the most relevant parts of the input. This design allows the model to focus on different parts of the sequence depending on the task and context. The key advantage of QKV is its ability to support self-attention and multi-head mechanisms efficiently, enabling parallel computation and capturing both local and global dependencies in a highly flexible manner.

- Query (Q)
- Key (K)
- Value (V)

These vectors are obtained by linear transformations:

$$Q = XW^Q, \quad K = XW^K, \quad V = XW^V,$$

where:

- $X \in \mathbb{R}^{n \times d}$ is the input matrix,
- n is the sequence length,
- d is the model dimensionality,
- $W^Q, W^K, W^V \in \mathbb{R}^{d \times d_k}$ are learned projection matrices,
- d_k is the dimensionality of the projected space.

This separation allows each element to perform different roles—querying, being queried, and carrying content.

Scaled Dot-Product Attention

Computing the attention scores using the dot product of each query with every key. This step determines how much attention a particular input element should pay to the others. Mathematically, the raw attention score between query Q_i and key K_j is:

$$\text{score}(i, j) = \frac{Q_i \cdot K_j^\top}{\sqrt{d_k}},$$

This scaling prevents the dot products from becoming too large, which could lead to instability during training.

The scores are passed through a softmax to compute normalized attention weights:

$$\text{AttentionWeights}_{i,j} = \text{softmax}_j\left(\text{score}(i, j)\right),$$

This softmax ensures that the weights assigned to each key sum to 1 for a given query. These weights can be interpreted as the importance of each key (and thus its associated value) with respect to the current query.

The final attention output to compute is a weighted sum of the value vectors using the attention weights. For each query Q_i, the attention output is:

$$\text{Output}_i = \sum_{j=1}^{n} \text{AttentionWeights}_{i,j} \cdot V_j,$$

This mechanism allows a new representation for each input element, now enriched with contextual information from the rest of the sequence. The values V carry the actual content to be transferred, while the queries and keys determine which parts are most relevant.

This process is vectorized in practice as:

$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^\top}{\sqrt{d_k}}\right)V.$$

Multi-Head Attention

Attention mechanisms allow the use of multi-head attention, where multiple heads operates in a distinct subspace with separate Q, K, and V sets operate in parallel.

Each attention head has its own set of learned projection matrices and processes the input independently. This setup enables each head to specialize in capturing different types of relationships such as syntactic patterns, semantic meanings, or positional cues.

These distinctions emerge because each head operates in a different subspace, meaning that each head projects the input embeddings into its own learned lower-dimensional space using distinct weight matrices.

This means that even though all heads receive the same input sequence, they extract different types of features because their projections (the Q, K, and V matrices) are independently parameterized. As a result, one head might focus on short-range dependencies while another attends to long-range context, or one head might learn to emphasize spatial relationships while another prioritizes semantic content. This architectural design enables the model to disentangle and capture various relational aspects simultaneously, enriching the expressiveness and flexibility of the resulting representation. position between tokens while another might align similar color patterns across a visual grid.

After computing their respective attention outputs, the results of all heads are concatenated and passed through a linear transformation, allowing the model to combine the diverse relational insights into a unified representation that is both rich and context-aware.

This structure enables richer and more flexible representations by disentangling various types of relationships across the input sequence.

2D Relative Positional Bias Module

During the calculation of the attention mechanism a relative positional encoder bias (RPE) is added. In standard Transformers, absolute positional encoding gives each position a fixed representation. However, this doesn't tell the model 'how far apart' two tokens are, or whether one is to the left/right, above/below another. to address that, the relative positional encoding allows the model to learn how attention should vary based on the relative spatial distance between tokens

improving the model's ability to learn complex spatial relationships.

The RPE is implemented as a trainable bias matrix of shape $(S_{\text{grid}}, S_{\text{grid}})$, where S_{grid} is the total number of tokens in the 2D grid (i.e., after flattened an image or grid of pixels). This matrix contains the attention bias for each pair of positions in the grid.

To compute each entry in this bias matrix for token positions (i, j) and (i', j'), the model calculates the Manhattan distance:

$$d = |i - i'| + |j - j'|,$$

This distance reflects how far apart two grid positions are in a horizontal/vertical (non-diagonal) sense.

To incorporate **directional asymmetry**, different scaling slopes are used depending on whether the second token is 'to the left' or 'to the right' of the first one. These are defined as:

$$r_{\text{direction}} = \begin{cases} r_{\text{left}} & \text{if } j' < j \\ r_{\text{right}} & \text{otherwise,} \end{cases}$$

The final bias is then:

$$b_{(i,j),(i',j')} = d \cdot r_{\text{direction}},$$

This allows the model to learn that certain directions (e.g., tokens to the left) should be weighted differently during attention.

Incorporating Bias into Attention

During attention computation, the bias is added to the scaled dot-product scores:

$$\tilde{\text{Scores}}_{i,j} = \frac{Q_i \cdot K_j^\top}{\sqrt{d_k}} + B_{i,j},$$

The final attention weights become:

$$A_{i,j} = \frac{\exp(\tilde{\text{Scores}}_{i,j})}{\sum_{k=1}^{S} \exp(\tilde{\text{Scores}}_{i,k})},$$

- $\tilde{\text{Scores}}_{i,j}$ is the attention score between token i and token j, including the bias.
- $B_{i,j}$ is extracted from the full bias matrix to match the sequence length.

This approach improves the model's ability to reason over structured spatial layouts, as in grid-based tasks like ARC.

Feed-Forward Network (FFN)

After the attention outputs from the eight heads are computed, they are concatenated and passed through a linear layer known as the output projection. This layer integrates information from the different attention heads, creating a unified representation that combines the diverse relational features captured by each head.

Next, a dropout operation is applied to the projected attention output. Dropout is a regularization technique used to prevent overfitting. During training, each neuron's output has a probability p (e.g., 0.01 or 1%) of being dropped (i.e., temporarily set to zero). To maintain the expected output magnitude, the remaining activations are scaled by a factor of $1/(1-p)$.

$$\text{output}_i = \begin{cases} 0 & \text{with probability } p \\ \frac{x_i}{1-p} & \text{with probability } (1-p), \end{cases}$$

This technique discourages co-adaptation, where neurons become overly dependent on specific other neurons. Instead, each neuron is encouraged to learn features that are independently useful, resulting in a more robust model.

After dropout, the output is passed through a normalization layer, typically Layer Normalization. This helps stabilize training, reduce sensitivity to parameter initialization, and speed up convergence by maintaining consistent activation distributions across layers.

Loss Function

Choosing the appropriate loss function is critical for model performance and training stability. Several options are possible, each with distinct trade-offs:

Mean Squared Error (MSE)

The Mean Squared Error is defined as:

$$\text{MSE} = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2,$$

While MSE penalizes the squared difference between the predicted and true values, it treats each prediction independently. This is suboptimal for structured tasks like grid predictions which is the current scenario, where spatial or structural coherence matters. MSE does not capture relationships or dependencies between pixels.

Cosine Similarity

Cosine similarity measures the alignment (direction) of two vectors:

$$\text{CosineSimilarity}(y, \hat{y}) = \frac{y \cdot \hat{y}}{\|y\| \|\hat{y}\|},$$

This metric focuses on the angle between vectors rather than their magnitude. While it captures directional accuracy, it is scale-invariant and may not penalize large magnitude errors. As a result, the model might produce outputs that are directionally correct but structurally inaccurate.

Cross-Entropy Loss

Given that the grid contains a fixed set of possible token values (0–12), the prediction task is naturally framed as a classification problem. Each token represents a semantic combination of color, position, and attention-derived information. In this setting, Cross-Entropy Loss is the most appropriate:

$$\mathcal{L}_{\text{CE}} = -\sum_{i=1}^{n} y_i \log(\hat{y}_i).$$

Cross-entropy evaluates the divergence between the predicted probability distribution and the one-hot encoded true label. This loss is well-suited for token-level prediction in a ViT model, where the model must choose the most probable token (e.g., pixel class) at each position. It leverages the combined semantic and positional context encoded into the token representations.

Q-Learning Fine-Tuning

Reinforcement Learning (RL)

Reinforcement learning (RL) [3] is a branch of machine learning concerned with how agents learn to make decisions through interactions with an environment in order to maximize cumulative rewards. At each time step, the agent takes an action, receives a reward, and transitions to a new state. Over time, it updates its policy to improve future performance.

This interaction is modeled as a Markov Decision Process (MDP) a mathematical framework for modeling decision-making situations where outcomes are partly random and partly under the control of an agent and is defined by:

$$\langle \mathcal{S}, \mathcal{A}, p, r, \gamma \rangle.$$

[3] Richard S. Sutton. Learning to predict by the methods of temporal differences. *Machine Learning*, 1988. DOI: 10.1007/BF00115009

Where:

- $\mathcal{S}$: set of states (All possible situations the agent might encounter),
- $\mathcal{A}(s)$: set of available actions in state s (Actions that the agent can choose),
- $p(s'|s,a)$: transition probability of moving to state s' after taking action a in state s,
- $r(s,a)$: reward received after performing action a in state s,
- $\gamma \in [0,1]$: discount factor: value between 0 and 1 that controls the importance of future rewards compared to immediate ones.

The objective is to learn an optimal policy π^* that maximizes the expected return:

$$G_t = \sum_{k=0}^{\infty} \gamma^k r_{t+k+1},$$

Key functions in RL include:

- **State-value function** which measures the expected return starting from state s and following policy π.:

$$V^{\pi}(s) = \mathbb{E}_{\pi}[G_t \mid s_t = s],$$

- **Action-value function (Q-value)** which measures the expected return starting from state s, taking action a, and then following policy π.:

$$Q^{\pi}(s,a) = \mathbb{E}_{\pi}[G_t \mid s_t = s, a_t = a].$$

Q-Learning: Value-Based Learning

Q-learning [4] is an off-policy, value-based reinforcement learning algorithm that learns the optimal action-value function $Q^*(s,a)$. It uses the following update rule based on the Bellman optimality equation:

$$Q_{t+1}(s_t,a_t) = Q_t(s_t,a_t) + \alpha \left[r_{t+1} + \gamma \max_{a'} Q_t(s_{t+1},a') - Q_t(s_t,a_t) \right],$$

Where α is the learning rate. This iteration converges under certain conditions to $Q_*(s,a)$.

In Q-learning, the term "off-policy" refers to how the algorithm learns. There is two types of policies: the behavior policy is the strategy the agent actually uses to explore the environment and collect experiences (i.e., sequences of state, action, reward, next state). The target policy is

[4] C. J. C. H. Watkins. *Learning from Delayed Rewards*. PhD thesis, University of Cambridge, 1989

the policy that the agent is trying to learn and improve, with the goal of it becoming the optimal policy.

Off-policy algorithm learns about the target policy (which aims to be optimal) that is different from the policy it uses to generate actions and gather data (the behavior policy) it directly attempts to learn the optimal action-value function, denoted as $Q^*(s, a)$, which represents the maximum expected future reward achievable by taking action a in state s and then following the optimal policy thereafter. The update rule for Q-learning involves a term $\max_{a'} Q(s', a')$, which estimates the value of the best possible action in the next state s'. This "max" operator assumes that in the next state, the agent will take the action that maximizes the Q-value, which corresponds to the greedy (optimal) policy. The actual action taken to transition from state s to s' could have been chosen by a completely different, behavior policy (e.g., an epsilon-greedy policy that sometimes picks random actions). Q-learning still uses this experience to update its estimate of $Q^*(s, a)$ by considering what the optimal action would have been in state s'. This decoupling allows Q-learning to learn about the optimal way to behave even while it might be behaving sub-optimally or exploratory to gather diverse experiences.

The Bellman optimality equation [5] characterizes the maximum expected return achievable from any state (or state-action pair) under the optimal policy π^*. It provides a recursive relationship that defines the value of a state or state-action in terms of the values of successor states.

[5] Richard Bellman. *Dynamic Programming*. Princeton University Press, Princeton, NJ, 1957. ISBN 9780691079516

Deep Q-Learning: Function Approximation

For problems with large or continuous state spaces, maintaining a Q-table becomes impractical. Deep Q-Networks (DQNs) address this by approximating $Q^*(s, a)$ using a neural network $Q_\theta(s, a)$ parameterized by weights θ.

The network is trained to minimize the loss:

$$\mathcal{L}(\theta) = \frac{1}{2}\left(y^{\text{DQN}} - Q_\theta(s, a)\right)^2.$$

Where the target value is computed using a separate target network:

$$y^{\text{DQN}} = r + \gamma \max_{a'} Q_{\theta^-}(s', a').$$

To improve training stability, the DQN approach introduced by Mnih et al. (2015) [6] uses the following techniques:

- **Target network** Q_{θ^-}: a copy of the Q-network that is updated less frequently to provide stable targets.

[6] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A. Rusu, Joel Veness, Marc G. Bellemare, Alex Graves, Martin Riedmiller, Andreas K. Fidjeland, Georg Ostrovski, Stig Petersen, Charles Beattie, Amir Sadik, Ioannis Antonoglou, Helen King, Dharshan Kumaran, Daan Wierstra, Shane Legg, and Demis Hassabis. Human-level control through deep reinforcement learning. *Nature*, 2015. URL https://www.nature.com/articles/nature14236

- **Experience replay buffer**: stores past experiences (s, a, r, s') and samples mini-batches randomly to break correlations in the data.
- **ϵ-greedy policy**: balances exploration and exploitation by choosing a random action with probability ϵ, and the best-known action otherwise.

These innovations allow deep Q-learning to effectively scale to complex environments such as the ARC Benchmark.

Implementation in ViTARC-RL

The learning environment behaves as a contextual bandit, since each episode consists of a single step. The architecture is summarized as follows:

RL Concept	**Implementation in the script**
State s	Token sequence of the ARC grid (e.g., `state_seq`)
Action a	Replacing the last token with a color $a \in \{0, \ldots, 12\}$
Reward r	1.0 if action == target color; 0.0 otherwise
Approximator Q_θ	Linear `q_head` layer applied over frozen transformer embeddings
Target network	`target_q_head`, updated via `soft_update`
Experience buffer	`ReplayBuffer` with capacity 10,000
ϵ-exploration	`select_action()` and `anneal_epsilon()`
Bellman loss	`F.mse_loss(q_s_t, target)`

Because the episode terminates immediately (`done=True`), the bootstrap term

$$\gamma \max_{a'} Q_{\theta^-}(s', a'),$$

Vanishes, and the algorithm *reduces to supervised regression toward the 0/1 reward*—but it retains the DQN machinery so that the same scaffold can be used if multiple steps are later predicted.

Purpose of RL Fine-tuning

Adding a reinforcement learning (RL) stage after supervised training allows the model to shift its focus from simply replicating pixel-level targets—a proxy objective optimized through cross-entropy—to directly maximizing task-level success, such as selecting the correct final color. This is achieved through a bandit-style Q-learning fine-tuning step, which explicitly reinforces correct end-of-sequence decisions.

Crucially, by freezing nearly all of the model's parameters and only updating the final linear layer, the approach preserves the spatial reasoning learned during the supervised phase.

4 Experimental Results

Datasets

For this work, we used the public portion of the ARC dataset, which consists of 400 tasks. Each task is divided into train and test subsets, composed of multiple examples that vary in number from task to task.

Due to computational constraints, we selected only the first ten tasks from the database. Using a Domain-Specific Language (DSL), one thousand examples were generated for each of these tasks [1]. From these generated examples, we created four training subsets and one test subset per task. The training subsets consisted of 1, 10, 100, and 900 examples, respectively, while the test subset contained 100 examples.

[1] Michael Hodel. Domain specific language for the abstraction and reasoning corpus (arc-dsl). https://github.com/michaelhodel/arc-dsl, 2023

This setup allowed us to measure the impact of training sample size on the total training epochs and to assess the effects of sample size on test performance.

To measure performance we use two main metrics: Pixel Accuracy and Full Accuracy per task.

The pixel accuracy measures how many individual pixels are predicted correctly across all sequences.

$$\text{correct\_pixels} = \sum_{i,j} (\hat{y}_{i,j} = y_{i,j}),$$

where:

$y_{i,j}$ is the target value at position (i, j),

Total number of elements:

$$\text{total\_pixels} = N \times L,$$

where:

N is the number of grids and L is the number of elements per grid.

Compute pixel accuracy:

$$\text{pixel\_accuracy} = \frac{\text{correct\_pixels}}{\text{total\_pixels}},$$

Full Accuracy checks if the entire predicted sequence matches the target sequence.

$$\text{full\_match}_i = \forall j, \hat{y}_{i,j} = y_{i,j},$$

where $\forall j$ means "for all elements j in sequence i".

$$\text{full\_accuracy} = \frac{1}{N} \sum_i \text{full\_match}_i.$$

MLP Baseline Performance

Architecture

The MLP baseline model consists of the following architecture:

- Input layer: 931 neurons
- Hidden layers: 4096 → 4096 → 2048 → 2048 → 1024 → 1024 neurons
- Output layer: 931 neurons
- All hidden layers use ReLU activation
- Total trainable parameters: 54,106,019

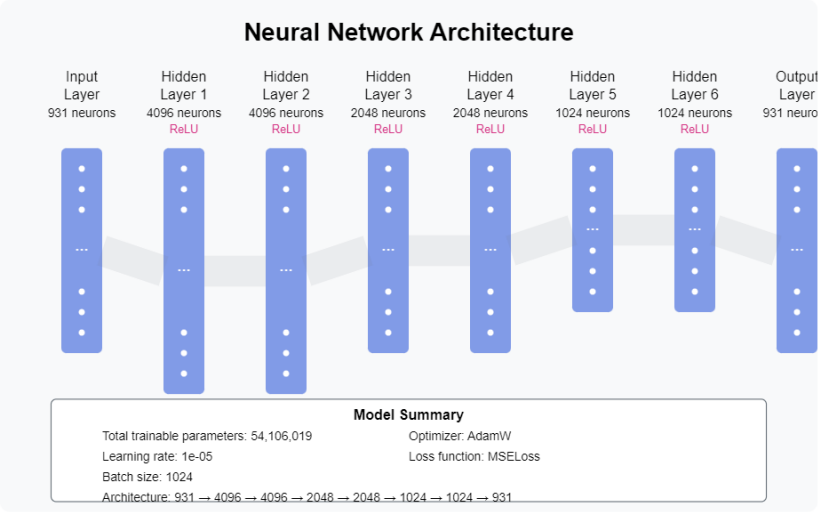


Figure 4.1: MLP Model Representation

The MLP baseline model configuration:

- Learning Rate: 1e-05
- Batch size: 1024
- Optimizer: AdamW
- Loss function: MSELoss

Training Performance

Pixel and Full accuracy per task, by training size

Pixel accuracy measures the proportion of individual prediction elements (such as pixels or tokens) that are correctly classified, full accuracy assesses whether the entire prediction matches the target exactly, requiring all elements in an example to be correct for it to count as accurate. While pixel accuracy is less sensitive to small errors and can remain high even if a prediction is only partially correct, full accuracy is a stricter metric that reflects the model's ability to produce completely accurate outputs.

As the number of training examples increases, the model requires more epochs to achieve 100% pixel accuracy. For instance, with 1 example, the model needed 2,000 epochs; with 10 examples, 7,000 epochs; with 100 examples, 17,500 epochs; and with 900 examples, more than 80,000 epochs (Figure 4.2). This reflects the complexity of the problem's structure. Although the underlying logic behind each example is the same, and new examples should theoretically be easier to learn due to shared structural patterns, the observed increase in training time suggests otherwise. This indicates that the model is unable to effectively generalize shared information across examples and instead must learn new structures for each one. As a result, the model saturates and requires substantially more epochs to converge.

Full accuracy per training task is achieved at 100% in al task and in all training sizes (Figure 4.3) with the exception of 100 examples probably because missing more training epochs. This results will be of relevance when compared with ViT-RL as that model wasn't unable to achieve that level of accuracy during training.

Test Performance

Pixel and Full accuracy per task, by training size

From the pixel accuracy results, we observe that the model achieves its highest performance with just one training example, followed by a sharp drop, and then a gradual improvement as the number of examples increases up to 900. However, it never fully recovers, with only 30% of tasks showing better results at 900 examples than at one example (Figure 4.4). This pattern occurs because, with a single example, the neural network effectively memorizes the shape, and comparisons with other examples from the same task show overlapping empty (black) pixel regions, leading to deceptively high accuracy. As more examples are introduced, even the empty spaces no longer align, causing a performance drop. Nevertheless, by the time the model sees 900

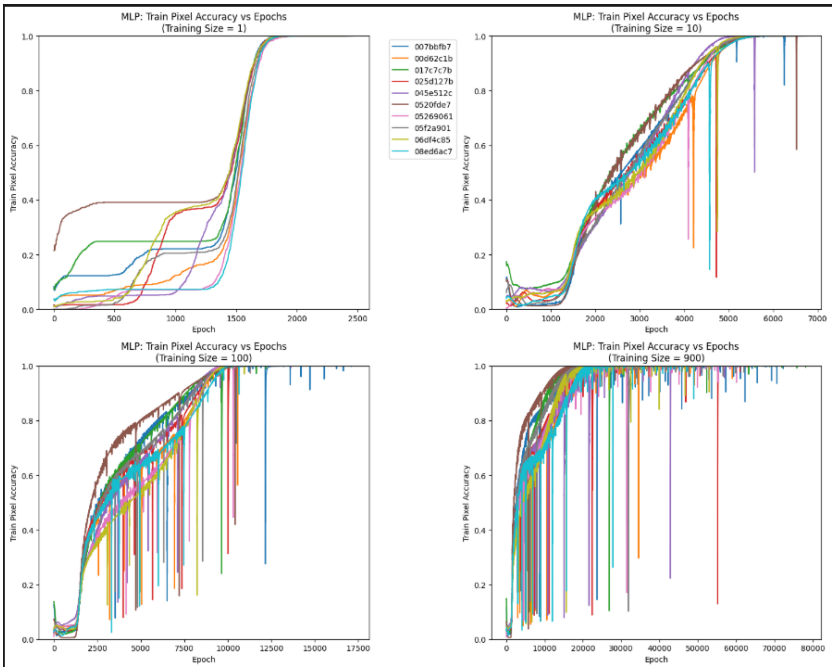


Figure 4.2: Pixel accuracy per task, by training size

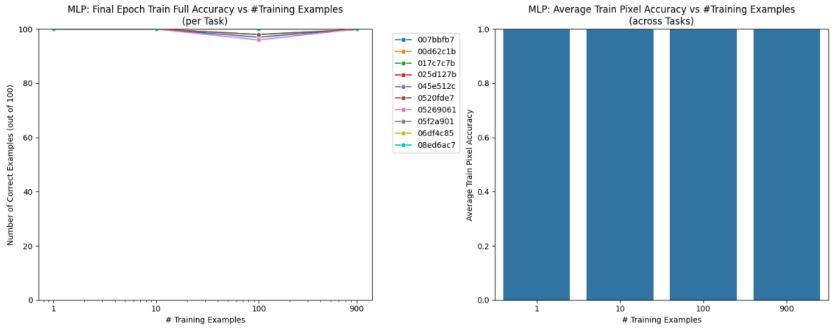


Figure 4.3: Full accuracy per task and average pixel accuracy

examples, it begins to learn underlying structural patterns rather than relying on overlap, resulting in some improvement. This phenomenon is also evident in the full accuracy results: high pixel accuracy does not necessarily correspond to fully correct predictions (e.g., with 1 example), while a larger number of examples—despite lower pixel accuracy—can lead to a few completely correct predictions, as seen in task '007bbfb7', where 8 out of 100 samples were correctly predicted.

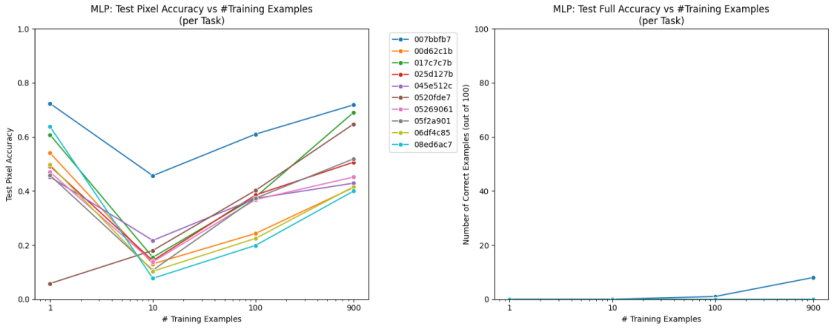


Figure 4.4: Pixel and Full accuracy per task

ViT-RL Model Architecture and Training Configuration

Architecture

The **ViT-RL** model consists of a transformer-based encoder-decoder architecture:

- **Encoder:** 6 layers, each with self-attention (qkv_proj, out_proj) and feedforward (ffn) sub layers
- **Decoder:** 6 layers, each with self-attention, cross-attention, and feedforward sub layers
- **Hidden size:** 128
- **Output layer:** `Linear(128, 13)`
- **Total trainable parameters:** 398,221

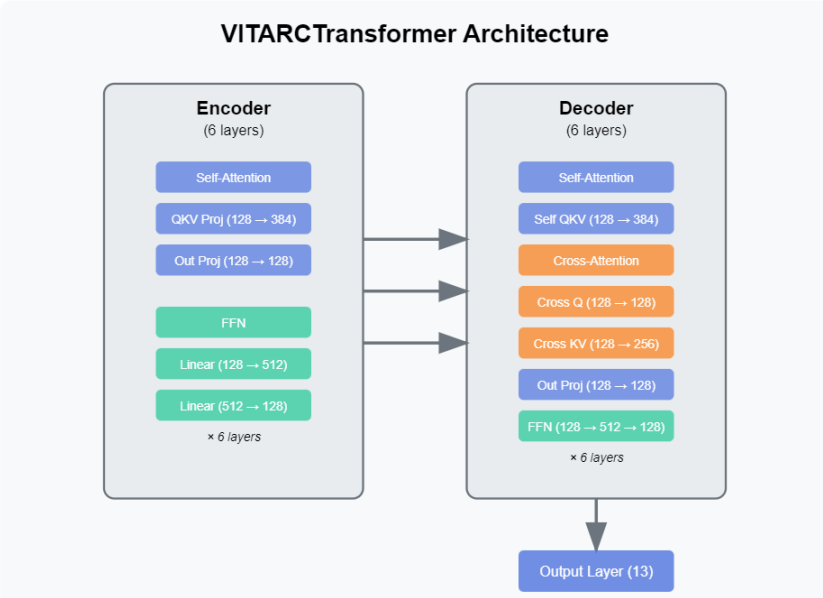
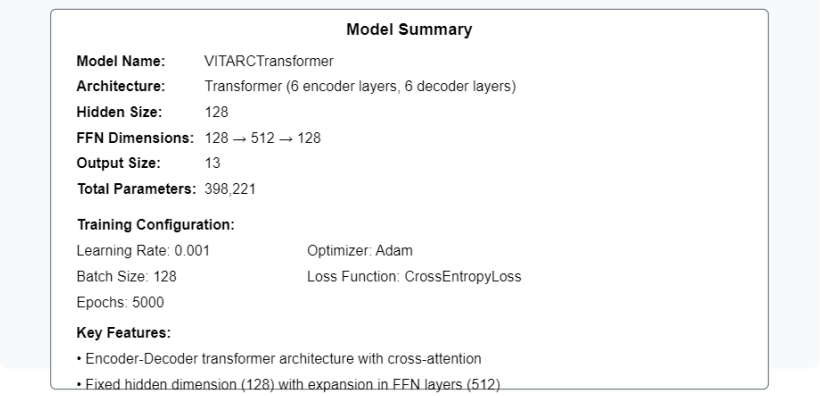


Figure 4.5: Model Architecture ViT-RL



Model Summary

Model Name: VITARCTransformer
Architecture: Transformer (6 encoder layers, 6 decoder layers)
Hidden Size: 128
FFN Dimensions: 128 → 512 → 128
Output Size: 13
Total Parameters: 398,221

Training Configuration:

Learning Rate: 0.001 Optimizer: Adam
Batch Size: 128 Loss Function: CrossEntropyLoss
Epochs: 5000

Key Features:

• Encoder-Decoder transformer architecture with cross-attention
• Fixed hidden dimension (128) with expansion in FFN layers (512)

Figure 4.6: Model Architecture ViT-RL

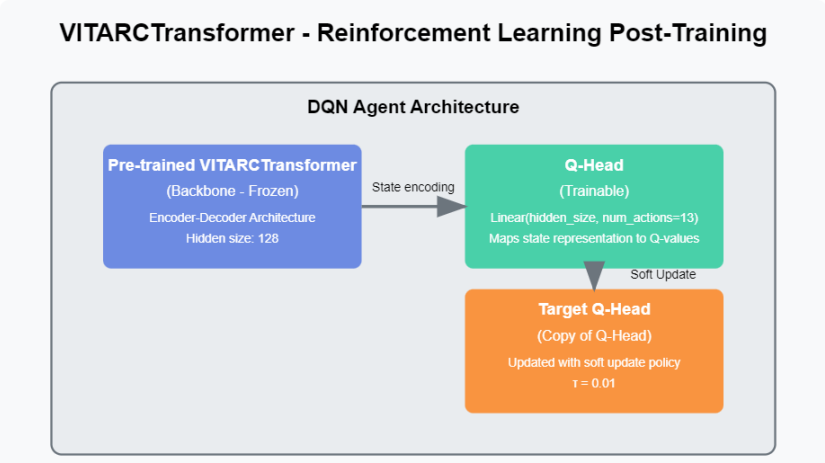


Figure 4.7: Model Architecture ViT-RL

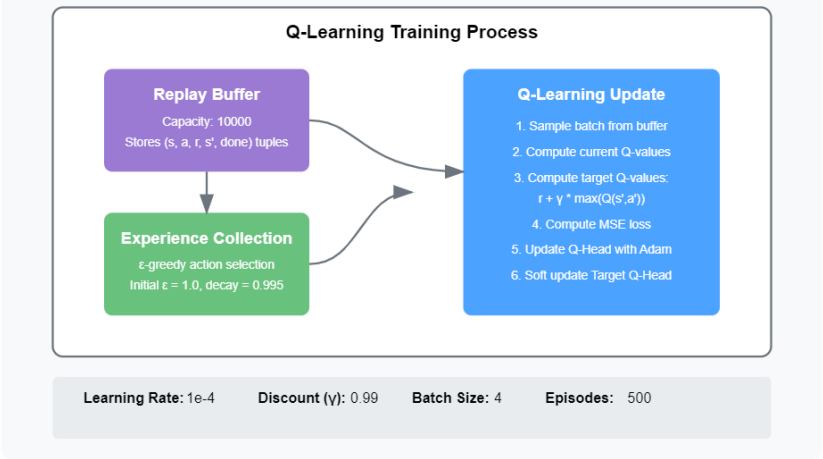


Figure 4.8: Model Architecture ViT-RL

4.0.1 Training Configuration

- **Learning rate:** 0.001
- **Batch size:** 128
- **Optimizer:** Adam
- **Loss function:** `CrossEntropyLoss`
- **Number of epochs:** 5000
- **Device:** `cuda`

Training Performance

Pixel and Full accuracy per task, by training size

Similar to the MLP, observations show that as the number of training examples increases, the number of epochs required also grows (Figure 4.9). However, we noticed that significantly fewer epochs are needed when training with only 1 example the model required 800 epochs in this case. With 10 examples, it needed 1,000 epochs; with 100 examples, 2,000 epochs; and with 900 examples, more than 5,000 epochs. This is likely due to the fact that the proposed model incorporates additional information that can be transferred to the neural network (NN).

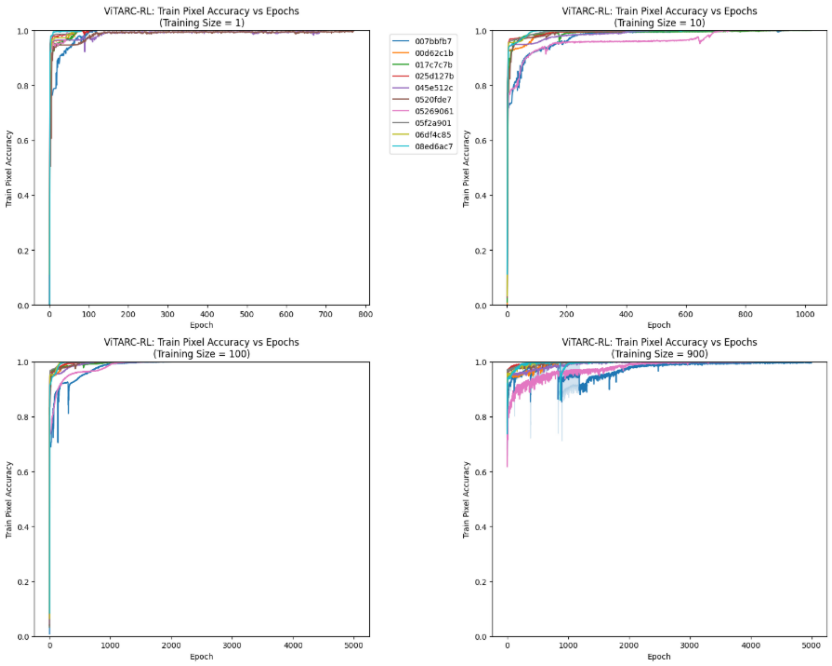


Figure 4.9: ViT-RL Train Pixel Accuracy by number of epochs

While the MLP model achieved nearly 100% accuracy in both Pixel and Full accuracy on the training set, we observed that ViT-RL reached 100% Pixel accuracy but was not able to fully predict all training examples correctly in terms of Full accuracy. Its Full accuracy ranged between 60% and 80% (Figure 4.10). This issue will be discussed further; however, the assumption that better training performance translates to better test performance does not hold in this case. Despite ViT-RL showing significantly worse Full accuracy during training compared to the MLP, it achieved much better performance on the test set, as we will demonstrate in the following section.

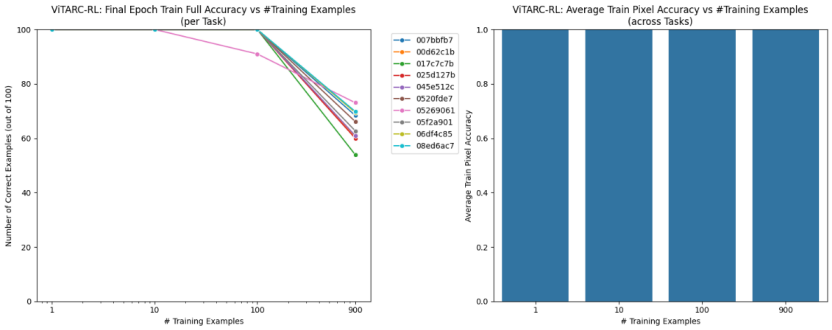


Figure 4.10: ViT-RL Pixel and Full Accuracy by training examples

Test Performance

Pixel and Full accuracy per task, by training size

As mentioned earlier, even though the training results for ViT-RL with 900 examples were worse than those of the MLP, ViT-RL significantly outperformed the MLP on the test set across all training sizes. We also observed better generalization in ViT-RL as the number of examples increased, unlike the MLP, which exhibited a 'U'-shaped performance curve—excelling in some tasks when memorizing a single example. This suggests that ViT-RL demonstrates a stronger capacity to capture the underlying logic of the task, even with just one training example, rather than simply memorizing or mapping input-output pairs.

Although ViT-RL was not able to achieve perfect Pixel accuracy on the 900-example training set, its performance on the test set was the best among all configurations, with results ranging between 90% and 100% accuracy. This indicates that, despite lower training accuracy, the model generalizes significantly better.

In the case of the MLP, we observed that higher Pixel accuracy does not necessarily correlate with better Full accuracy. As previously discussed, this is likely due to the model improving its Pixel accuracy by correctly predicting empty or "void" pixels in the grid. In contrast,

for ViT-RL, improvements in Pixel accuracy were also reflected in better Full accuracy, indicating a deeper understanding of the task. Notably, ViT-RL successfully predicted the entire grid in 6 out of the 10 tasks, and in task '007bbfb7', it was able to correctly predict 27 test cases (Figure 4.11)(Figure 4.12).

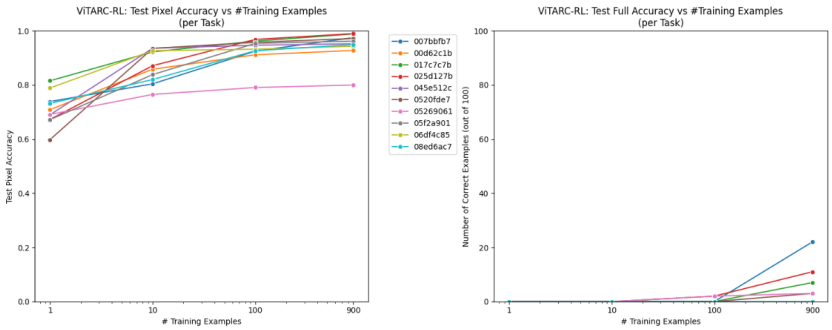


Figure 4.11: ViT-RL Pixel and Full Accuracy by test examples

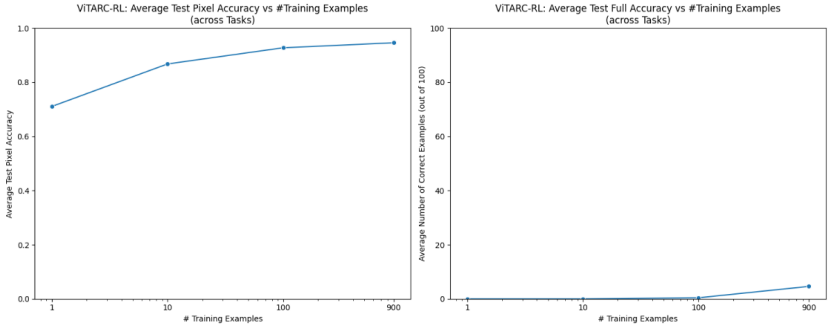


Figure 4.12: Average ViT-RL Pixel and Full Accuracy

Comparative Performance: MLP vs ViT-RL

Average Test Pixel Accuracy and Full Accuracy

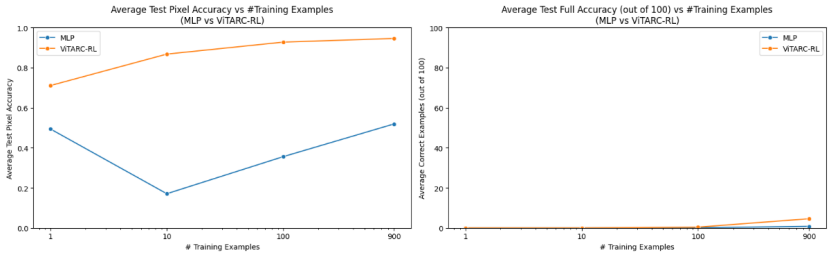


Figure 4.13: Average Test Pixel Accuracy and Full Accuracy

FIGURE 4.13 We can observe that show consistent improvement in

pixel accuracy as the number of training examples per task increases, while MLP have the best results with 1 and 900 examples, reflecting the characteristic "U" shape for memorization in the 1 example scenario.

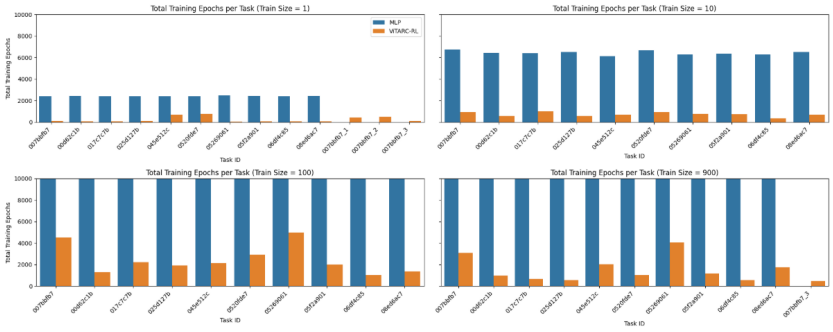


Figure 4.14: MLP vs ViT-RL training epochs by task

FIGURE 4.14 MLP consistently requires more epochs to converge, with 900 examples MLP requires over 25,000 epochs while ViT-RL is always under 5,000 epochs. This not necessarily means ViT-RL is more efficient as it is a bigger more that requires more time per epochs, however it have the parallelization advantage. In term of learning it is more efficient as much more information is imputed to the model than in the MLP.

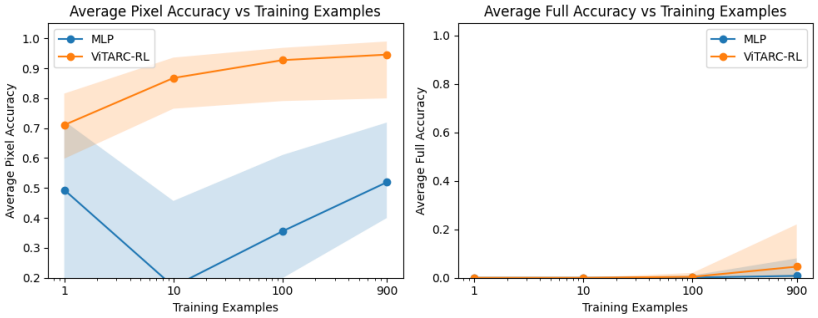


Figure 4.15: MLP vs ViT-RL training pixel and full accuracy by training examples

FIGURE 4.15 ViT-RL have tighter confidence intervals across training sizes, compare to MLP that have wider intervals which can be interpreted as higher variability and unstable generalization from MLP compare to ViT-RL

Comparative Analysis of ViT-RL Variants (Full, Similar, Diverse)

From previous experiments we can infer that more examples have more information which lead to find the most efficient way to capture that

information. Core-set selection [2] is the process of selecting a small, representative subset from a larger dataset such that training a model on this subset performs almost as well as training on the full dataset. By choose information rich subset of the data can rival sometimes even surpass the performance obtained from the full training set. Historically, this idea emerged from geometric covering arguments: if the inputs lie on a manifold $\mathcal{M}$, one can seek a subset S that minimizes the furthest distance between any point and its nearest representative. Formally, the k-center objective is to choose $S \subseteq 1, \ldots, N$ with $|S| = k$ that solves

$$\min_{S} \max_{j \in \{1,\ldots,N\}} \min_{i \in S} \|\mathbf{z}_i - \mathbf{z}_j\|_2,$$

where $\mathbf{z}\_i$ denotes a vector of example i. Although the exact problem is NP-hard, a simple greedy "farthest-point" heuristic offers a 2-approximation and is widely used in practice. Sener and Savarese [3] (2018) translated this geometric criterion into a generalization-error bound for deep networks, showing that the radius of the cover acts as an upper bound on the gap between training and test risk.

We present a diverse 90 examples subset that approximates k-center solution: by repeatedly picking the example whose Euclidean distance to the current subset is largest, shrink the uncovered radius of the training manifold and therefore preserve most of the mutual information between data and labels. In information theory [4] , each additional diverse sample increases the mutual information $\mathcal{I}(X\_S; Y)$ because it reduces the conditional entropy $\mathcal{H}(Y \mid X\_S)$ in a distinct region of the feature space. Conversely, the similar 90 example subset formed by selecting the densest cluster contains highly redundant points that share information is used to performance comparative analysis.

When train ViT-RL on the diverse subset, the model receives broad coverage of pattern variations even though it sees only one tenth of the original 900 examples. Because the Euclidean distance is computed the vector examples, the selection emphasize raw pixel distance.

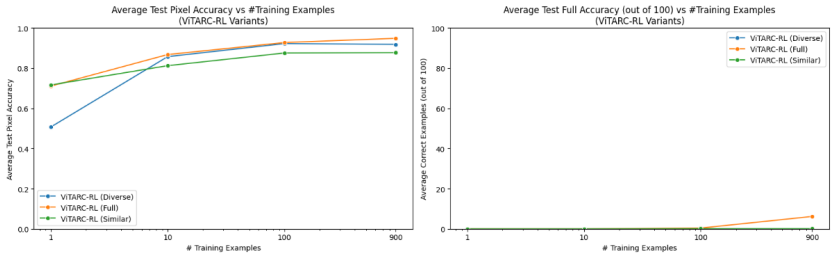


Figure 4.16: ViT-RL full diverse and similar comparative on pixel and full accuracy by training examples

FIGURE 4.16 Observe that the accuracy improve with all dataset, even

[2] Baharan Mirzasoleiman, Jeff Bilmes, and Jure Leskovec. Coresets for data-efficient training of machine learning models. In *Proceedings of the 37th International Conference on Machine Learning*, 2020. URL https://cs.stanford.edu/people/jure/pubs/craig-icml20.pdf

[3] Ozan Sener and Silvio Savarese. Active learning for convolutional neural networks: A core-set approach. In *International Conference on Learning Representations (ICLR)*, 2018. URL https://openreview.net/forum?id=H1aIuk-RW

[4] Claude E. Shannon. A mathematical theory of communication. *Bell System Technical Journal*, 27, 1948

at one example Similar is able to outperform Diverse. In full accuracy neither Similar all Diverse where able to predict, more testing is needed to find at what size Diverse dataset could get results similar to the full dataset.

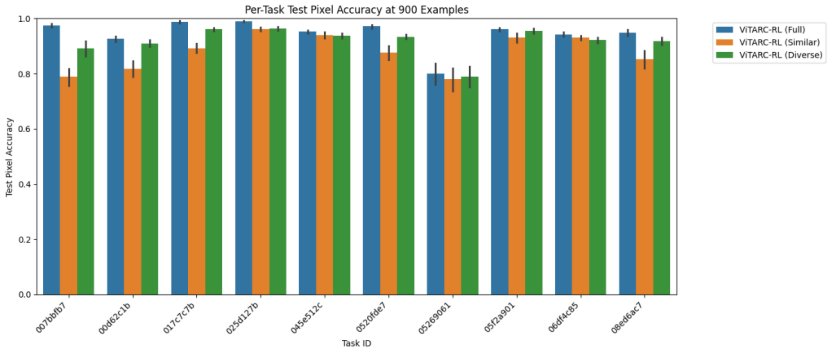


Figure 4.17: ViT-RL full diverse and similar comparative on pixel and full accuracy by training examples

FIGURE 4.17 Not all task have the same level of complexity and some may required information outside of the dataset, therefore we can observe that full dataset constantly outperforms however there is task where Similar outperform Diverse like; o6df4c85 and 045e512c.

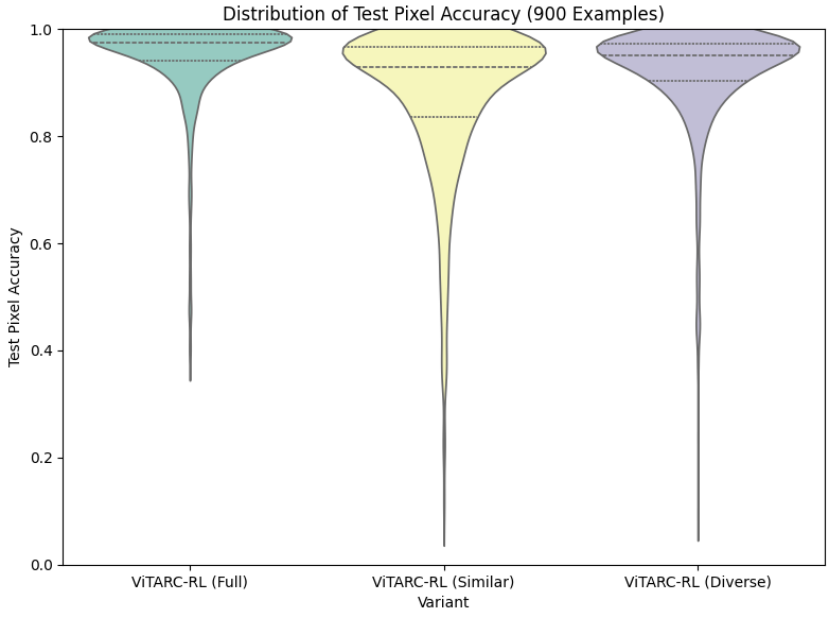


Figure 4.18: ViT-RL full diverse and similar comparative on pixel and full accuracy by training examples

FIGURE 4.18 All three dateset show high pixel accuracy, however Similar has the widest spread and more outliers with lower

performance, Diverse have more compact distribution and lower variance which means more consistent performance across task.

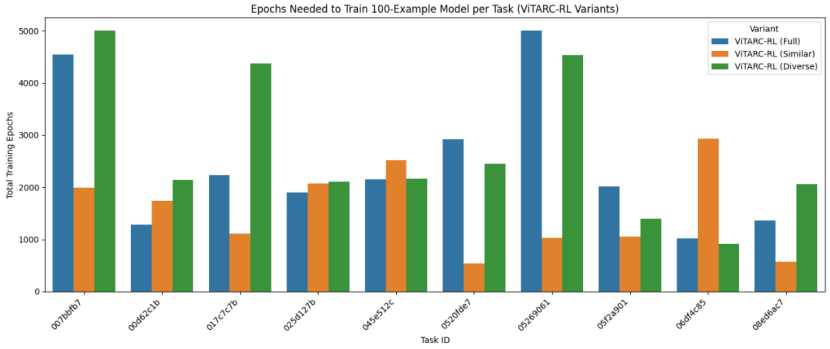


Figure 4.19: ViT-RL full diverse and similar comparative on pixel and full accuracy by training examples

FIGURE 4.19 per task results are mixed as each model have at least one task where was the training set that required the most epochs to converge, however overall Similar required less epochs to converge. At this dataset size the efficiency on Diverse dataset does not justify the lost on performance specially when evaluated with full accuracy.

5 Conclusions

This thesis set out to examine whether ViTARC-RL could bridge the long-standing performance gap on the Abstraction and Reasoning Corpus (ARC). We began by analyzing why state-of-the-art (SOTA) networks, despite excelling on benchmarks such as ImageNet and GLUE, struggle with ARC's requirement for one-shot abstraction and rule induction. Three primary limitations emerged: loss of spatial structure when inputs are flattened, an overreliance on color tokens rather than positional cues, and the absence of an explicit search mechanism for sequential decision-making.

We discovered that preserving information representation significantly impacts performance. Utilizing 2D absolute and relative positional encodings combined with the PEmixer retains both color and geometric information. This approach resulted in a test pixel accuracy of **0.869** across the ten ARC tasks tested, representing a 35 points improvement over the CNN baseline and a 65 points improvement over the MLP baseline. Although the MLP achieved nearly perfect training accuracy, it displayed a U-shaped test accuracy curve. In contrast, the test accuracy of ViTARC-RL increased gradually with additional examples and surpassed the MLP even when trained with only a single example per task. This supports the claim that structured attention mechanisms facilitate the discovery of underlying functions rather than merely memorizing pixel layouts.

It is also noteworthy that freezing the transformer and updating only the Q-head improved full-grid correctness without negatively affecting pixel precision; F1 scores increased from 0.050 to 0.281 compared to a pure ViT. These results suggest that even a single-step reinforcement learning (RL) scaffold can significantly enhance network accuracy.

When training with a 10% "diverse" subset, we observed that the model approaches the performance of training with the full 900-example set in half the epochs. This finding aligns with literature on geometric core-sets, although it still trails slightly behind the full dataset in absolute accuracy.

Due to computational constraints, experiments were limited to only ten ARC tasks, an there is a need for broader validation. Additionally,

the RL phase implemented is currently a single step contextual bandit. Multi step planning methods, such as Dyna-Q or tree search, could potentially yield improved results. Achieving full accuracy remains an objective, with the current model reaching approximately 80% of human performance.

For future research, the model should ideally be evaluated on the complete set of 400 public ARC tasks and submitted to the private ARC AGI leader board to comprehensively measure progress. Moreover, integrating a symbolic Domain-Specific Language (DSL) as the action space for the Q-head allowing RL to construct higher level functions instead of single pixel actions could substantially enhance model performance.

The Vision Transformer with Reinforcement Learning (ViT-RL) approach does not yet fully solve ARC but narrows the gap, highlighting specific areas where current neural models fall short. By maintaining spatial semantics and coupling them with explicit search mechanisms, ViT-RL represents a concrete advancement towards the broader goal of enabling machines to reason, rather than merely recognize patterns. We hope that the insights provided here will inspire continued systematic and interpretable advancements toward achieving general intelligence.

Bibliography

M. Andrews. Capturing sparks of abstraction for the arc challenge. https://arxiv.org/abs/2411.11206, 2024.

Richard Bellman. *Dynamic Programming*. Princeton University Press, Princeton, NJ, 1957. ISBN 9780691079516.

N. Carion, F. Massa, G. Synnaeve, N. Usunier, A. Kirillov, and S. Zagoruyko. End-to-end object detection with transformers. https://arxiv.org/abs/2005.12872, 2020.

F. Chollet, M. Knoop, G. Kamradt, and B. Landers. Arc prize 2024: Technical report. https://arxiv.org/abs/2412.04604, 2024.

George Cybenko. Approximation by superpositions of a sigmoidal function. *Mathematics of Control, Signals and Systems*, 1989.

A. Dosovitskiy, L. Beyer, A. Kolesnikov, D. Weissenborn, X. Zhai, T. Unterthiner, ..., and N. Houlsby. An image is worth 16x16 words: Transformers for image recognition at scale. https://arxiv.org/abs/2010.11929, 2020.

Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial nets. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2014.

Michael Hodel. Domain specific language for the abstraction and reasoning corpus (arc-dsl). https://github.com/michaelhodel/arc-dsl, 2023.

A. Johnson, W. K. Vong, B. M. Lake, and T. M. Gureckis. Fast and flexible: Human program induction in abstract reasoning tasks. https://arxiv.org/abs/2103.05823, 2021.

Kaggle. Arc prize 2024. https://www.kaggle.com/competitions/arc-prize-2024, 2024a.

Kaggle. Arc prize 2024. https://www.kaggle.com/competitions/arc-prize-2024, 2024b.

Kaggle. Discussion on arc prize 2024. https://www.kaggle.com/competitions/arc-prize-2024/discussion/545844, 2024c.

Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 1998.

W. Li, Y. Xu, S. Sanner, and E. B. Khalil. Tackling the abstraction and reasoning corpus with vision transformers: the importance of 2d representation, positions, and objects. https://arxiv.org/abs/2410.06405, 2024a.

W.-D. Li, K. Hu, C. Larsen, Y. Wu, S. Alford, C. Woo, ..., and K. Ellis. Combining induction and transduction for abstract reasoning. https://arxiv.org/abs/2411.02272, 2024b.

Baharan Mirzasoleiman, Jeff Bilmes, and Jure Leskovec. Coresets for data-efficient training of machine learning models. In *Proceedings of the 37th International Conference on Machine Learning*, 2020. URL https://cs.stanford.edu/people/jure/pubs/craig-icml20.pdf.

Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A. Rusu, Joel Veness, Marc G. Bellemare, Alex Graves, Martin Riedmiller, Andreas K. Fidjeland, Georg Ostrovski, Stig Petersen, Charles Beattie, Amir Sadik, Ioannis Antonoglou, Helen King, Dharshan Kumaran, Daan Wierstra, Shane Legg, and Demis Hassabis. Human-level control through deep reinforcement learning. *Nature*, 2015. URL https://www.nature.com/articles/nature14236.

R. Pfister and H. Jud. Understanding and benchmarking artificial intelligence: Openai's o3 is not agi. https://arxiv.org/abs/2501.07458, 2025.

Ozan Sener and Silvio Savarese. Active learning for convolutional neural networks: A core-set approach. In *International Conference on Learning Representations (ICLR)*, 2018. URL https://openreview.net/forum?id=H1aIuk-RW.

Claude E. Shannon. A mathematical theory of communication. *Bell System Technical Journal*, 27, 1948.

D. Silver, J. Schrittwieser, K. Simonyan, I. Antonoglou, A. Huang, A. Guez, ..., and D. Hassabis. Mastering chess and shogi by self-play with a general reinforcement learning algorithm. https://arxiv.org/abs/1712.01815, 2017.

K. Singhal and G. Shroff. Conceptsearch: Towards efficient program search using llms for abstraction and reasoning corpus (arc). https://arxiv.org/abs/2412.07322, 2024.

R. S. Sutton and A. G. Barto. *Reinforcement Learning: An Introduction*. MIT Press, 2018.

Richard S. Sutton. Learning to predict by the methods of temporal differences. *Machine Learning*, 1988. DOI: 10.1007/BF00115009.

UCI Machine Learning Repository. Iris dataset example. https://archive.ics.uci.edu/ml/datasets/iris, 1988.

A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin. Attention is all you need. https://arxiv.org/abs/1706.03762, 2017.

C. J. C. H. Watkins. *Learning from Delayed Rewards*. PhD thesis, University of Cambridge, 1989.

Index