Bridging Computational Foundations to Generative AI:

A Design-Based Framework for Progressive Creative Coding Education

submitted as a requirement for the degree of
Master of Computer Science 2026
Berlin, Germany, March 2026

by
M.A. Burak Kağan Yılmazer
(3121346)

SRH Berlin University of Applied Sciences

Primary Thesis Supervisor:	Dr. Kristian Rother
Associate Thesis Supervisor:	Dr.-Ing. Joel Dokmegang

Abstract

Most people who use DALL-E or Stable Diffusion cannot explain what happens between the text prompt and the output image. Creative coding courses stop at visual patterns; machine learning courses start at calculus. Nothing published bridges the two. Can a single curriculum cover that distance? Pixels2GenAI is a fifteen-module sequence built around NumPy arrays, spanning pixel manipulation, geometry, image transformations, fractals, simulation, noise functions, classical machine learning, animation, neural networks, TouchDesigner integration, and generative AI models. Modules alternate between two formats: Hands-On Discovery, which opens with a runnable sketch before adding theory, and Conceptual Deep-Dive, where notation and formalism come first. Nine people tested six of the fifteen modules during a one-day workshop in February 2026. Their programming backgrounds varied widely, from under a year to over a decade. Data came from a 24-item knowledge test (pre and post), NASA-TLX workload ratings after every module, open-ended exit tickets, and a short feedback form. Scores went up. The group mean rose from 15.1% to 50.5% ($W = 0.0$, $p = .008$, $d = 1.615$), and "I Don't Know" selections dropped from 74% to 24%. Workload was not constant across the six modules ($\chi^2(5) = 15.795$, $p = .008$); mental demand drove most of that variation. One finding stood out above the rest: participants who arrived with more programming experience sat much lower on cognitive load ($r_s = -.857$, $p = .007$), yet experience had almost nothing to do with how much they learned ($r_s = +.204$, $p = .629$). Beginners and experienced coders gained at comparable rates. The cost differed; the outcome did not. Hands-On Discovery modules were less demanding on most TLX dimensions, but M3 (Convolution), a Conceptual Deep-Dive session, produced the highest understanding rating of any module. Content type, not framework label, seems to determine which format works. Generative AI items on the test improved only modestly ($g = 0.271$), and 56% of those responses remained "I Don't Know" after instruction. Six modules were not enough to reach that far. Whether the full fifteen-module sequence closes the gap is a question for the next iteration.

Acknowledgements

My deepest gratitude goes to my primary supervisor, Dr. Kristian Rother, who shaped this thesis from its earliest spark. The entire curriculum builds on the pedagogical structure he developed at `academis.eu/numpy_graphics`, and his willingness to let me extend that foundation into generative AI territory made the project possible. He did far more than supervise. He helped organise the workshop, facilitated sessions on the day, secured the physical space, reviewed every module I developed, and connected me with people whose contributions strengthened the work at every turn. He also helped recruit participants when I had none. I cannot overstate what his guidance meant throughout this process.

I also thank my secondary supervisor, Dr.-Ing. Joel Dokmegang, for his support and feedback during the formal milestones of this thesis.

Nine volunteers committed a full day on 12 February 2026 to work through six modules, complete surveys, and write candid feedback. Their data made this thesis real. Without their time and honesty, the empirical core of this study would not exist.

I thank the open-source communities behind Processing, p5.js, and Python's scientific stack, whose tools underpin every exercise in the curriculum.

Finally, my family. They were far away, but never absent. Their unwavering belief in me, and their willingness to support me through every stage of this journey, gave me the time and opportunity to pursue this work fully. I could not have done this without them.

Contents

List of Figures

List of Tables

List of Abbreviations

AI Artificial Intelligence.

APA American Psychological Association.

API Application Programming Interface.

CDD Conceptual Deep-Dive.

CLIP Contrastive Language–Image Pre-training.

CLT Cognitive Load Theory.

CNN Convolutional Neural Network.

CPSES Creative Problem-Solving Self-Efficacy Scale.

DBR Design-Based Research.

DCGAN Deep Convolutional Generative Adversarial Network.

DDIM Denoising Diffusion Implicit Models.

DDPM Denoising Diffusion Probabilistic Models.

DiT Diffusion Transformer.

GAN Generative Adversarial Network.

GIF Graphics Interchange Format.

GPU Graphics Processing Unit.

HOD Hands-On Discovery.

KR-20 Kuder–Richardson Formula 20.

MIDI Musical Instrument Digital Interface.

ML Machine Learning.

MSE Mean Squared Error.

NASA-TLX NASA Task Load Index.

ONNX Open Neural Network Exchange.

OSC Open Sound Control.

PACL Progressive Accumulation of Conceptual Layers.

PBS Project-Based Synthesis.

PCA Principal Component Analysis.

RGB Red, Green, Blue.

RQ Research Question.

SVM Support Vector Machine.

t-SNE t-Distributed Stochastic Neighbor Embedding.

UMAP Uniform Manifold Approximation and Projection.

VAE Variational Autoencoder.

VQ-GAN Vector Quantized Generative Adversarial Network.

VQ-VAE Vector Quantized Variational Autoencoder.

XOR Exclusive OR.

ZPD Zone of Proximal Development.

1. Introduction

A learner opens a Python script, sets three values in a NumPy array, and a coloured pixel appears on screen. Several modules later, the same learner trains a model that generates images from random noise. This thesis asks whether a structured sequence of programming exercises can carry someone from that first pixel to that generative model without losing them along the way. It presents a fifteen-module curriculum that teaches image-making through code, starting with basic colour operations and advancing step by step through pattern generation, image filters, recursive structures, simple neural networks, and finally image-generating AI. Nine participants, ranging from near-beginners to experienced programmers, tested six of the fifteen modules in a one-day workshop in February 2026. They took a knowledge test before and after, rated how demanding each module felt, and wrote open-ended reflections on what worked and what did not. Scores rose significantly across the group. The perceived difficulty of each module varied in ways the curriculum's structure could partly explain, and participants who arrived with more programming experience consistently reported lower difficulty. Figure 1.1 shows the landing page of the curriculum platform. The rest of this chapter sets out the context, the specific questions the study pursued, and the boundaries of what it can claim.

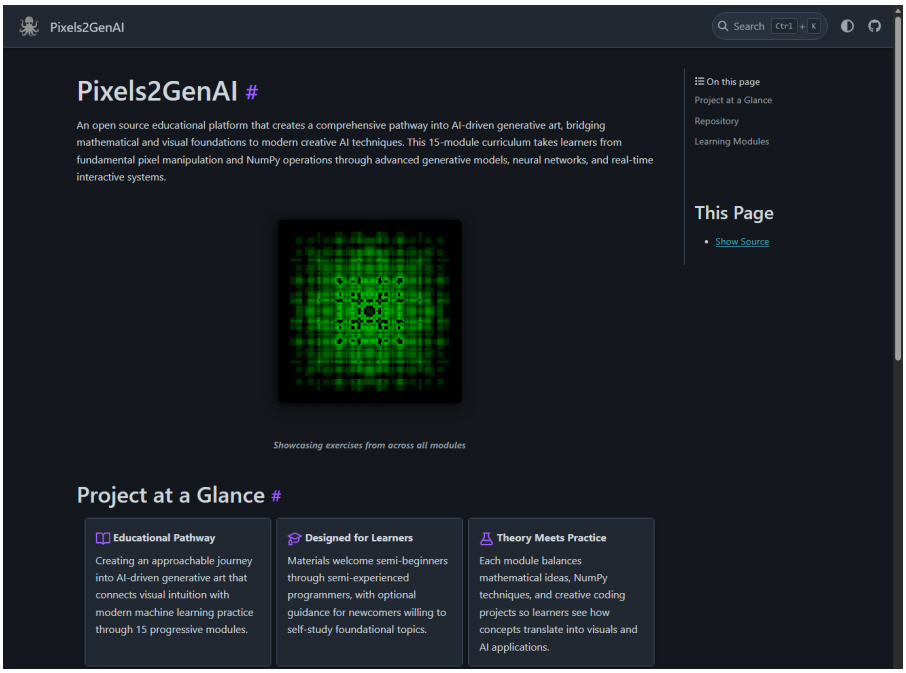


Figure 1.1.: Landing page of the Pixels2GenAI curriculum platform (https://burakkagann.github.io/Pixels2GenAI/, accessed March 2026).

1.1. Background and Context

DALL-E, Stable Diffusion, and Midjourney went from laboratory prototypes to widely adopted commercial products within a few years (Amankwah-Amoah et al. 2024), and the creative industries responded with a mixture of adoption and alarm (Amankwah-Amoah et al. 2024). Artists who had never written code began producing various types of visuals through natural-language prompts, studios wired AI pipelines into their production workflows, and universities added generative AI modules to syllabi that, a year earlier, had not mentioned diffusion models at all (Amankwah-Amoah et al. 2024). That speed surfaced a problem that slower technology cycles had kept hidden: most people who use generative AI have no way of understanding what it does. They type a prompt, an image appears, and everything in between stays in the dark. A spreadsheet user can, in principle, trace the logic of a formula. Tracing how a latent diffusion pipeline turns text into pixels requires probability theory, gradient-based optimization, and comfort with high-dimensional array operations. Few introductory courses cover any of that, let alone all of it in sequence.

Creative coding offers one piece of the answer. Processing, p5.js, and openFrameworks have spent over two decades showing that programming instruction works differently when the first output is a shape on screen rather than a string in a terminal (Reas et al. 2007). Papert (1980) made the foundational argument: a child who programs a turtle to draw a square engages with angle and distance in ways that a geometry textbook cannot replicate. Resnick et al. carried that thinking into digital media, arguing that effective learning environments combine easy entry, many creative directions, and room for expert-level work, a set of properties he labeled “low floor, wide walls, high ceiling.” (Resnick et al. 2009, p. 62) Those ideas shaped an entire ecosystem. Shiffman (2012) wrote a widely used creative coding textbook built on Processing; TouchDesigner brought node-based visual programming to live performance and installation art (Derivative 2023). What the movement proved is narrow but important: visual feedback pulls learners into computational thinking who would otherwise never start. Where it stops is equally clear. No creative coding platform extends its scaffolding to neural network architectures or generative model training.

The curricula that do cover generative models sit at the other end of the preparation spectrum. Established machine learning courses assume fluency and experience in linear algebra and calculus before the first lecture (Ng 2021). Tutorials on diffusion models and generative adversarial networks open with loss functions and gradient update rules (Ho et al. 2020; Goodfellow et al. 2014). A learner who can write a Processing sketch but has never inverted a matrix falls into the space between these two worlds, and nothing published catches them. Long et al. (2020) assembled a broad competency model for AI literacy, spanning five dimensions from recognition to societal reasoning, and showed that the construct cuts across age groups and disciplines. Sulmont et al. (2019) catalogued the pedagogical content knowledge that instructors need when teaching machine learning to non-experts, identifying persistent preconceptions (such as expecting deterministic outputs from probabilistic systems) that block understanding. Both contributions clarified what learners must eventually know.

Neither specified an instructional sequence for getting there, and neither engaged visual or creative practice as a design lever. Grover et al. (2013) surveyed computational thinking assessments and found them anchored to introductory programming constructs; nothing in that literature tracks the jump to neural network concepts.

The gap is strange, because the underlying concepts share more than the separate curricula let on. A convolution kernel is a small NumPy array that slides across an image. A perceptron takes a weighted sum of array elements and passes it through a threshold. A diffusion model adds Gaussian noise to an image array, then trains a network to reverse the corruption step by step. The mathematical objects barely change; what changes is how many of them interact at once. That framing comes from Cognitive Load Theory. Sweller (1988) argued that learning collapses when too many novel elements must be coordinated in working memory at the same time. Keep most elements familiar, introduce one or two new ones per step, and the load stays within bounds. Kirschner et al. (2006) put the corollary bluntly: unguided instruction in complex domains does not work, because novices lack the schemas that would let them manage element interactivity on their own. A curriculum that makes the shared array foundations explicit, building from pixels to kernels to convolution to feature maps to learnable weights, has a structural reason to believe it can hold intrinsic load at each transition.

Fifteen modules were built around that bet. Every module produces a visible output, and every module reuses data structures from the one before it. Two instructional frameworks, Hands-On Discovery and Conceptual Deep-Dive, package content differently depending on whether the topic is procedural or theory-heavy. A single-day workshop put six of those modules in front of nine participants whose programming backgrounds ranged from near-zero to over a decade.

1.2. Problem Statement

Four challenges fed into the design of the fifteen-module curriculum and the single-day workshop that put six of those modules to a first empirical test.

The first challenge is sequencing. Between a three-element NumPy array that colors a single pixel and a denoising network that generates a full image, dozens of intermediate concepts intervene: index arithmetic, convolution, activation functions, gradient updates, noise schedules. Published curricula skip most of those steps. Creative coding courses stop short of generative model training (Shiffman 2012; Reas et al. 2007). Machine learning courses assume the prerequisites are already in place (Ng 2021). No existing sequence threads a continuous path between these two endpoints, and learners who fall into the middle (comfortable with basic Python, unfamiliar with linear algebra) have nowhere to go. The competency model of Long et al. (2020) tells an instructor what a literate learner should eventually be able to do, but it does not specify an order for teaching those abilities; the pedagogical content knowledge framework of Sulmont et al. (2019) identifies the misconceptions that

block novice understanding of machine learning, yet offers no visual or creative pathway around them. The scaffolding problem, in short, is not that the individual concepts are too hard. Nobody has published a route linking them while keeping each step small enough to learn.

Keeping each step small enough is itself a non-trivial constraint. Too many unfamiliar elements held in working memory at the same time will stall learning; that is the central claim of Cognitive Load Theory (Sweller 1988). Generative AI stacks at least four knowledge domains on top of one another (programming, array mathematics, probability, and aesthetic judgment), so the count of interacting elements climbs fast. A convolution module asks a learner to manage kernel weights, sliding-window mechanics, and output-size arithmetic all at once. A diffusion module piles noise sampling, iterative denoising, and loss computation on top of those foundations. Without prior modules that have turned some of those elements into automated schemas, load at each new stage can blow past the ceiling that working memory sets (Kirschner et al. 2006). The theory itself is well established. What remains open is how to decompose and sequence specifically these topics so that load stays within bounds across a multi-module progression.

A related but separate difficulty is that different topics respond to different kinds of instruction. Fractal geometry benefits from open-ended exploration: learners change parameters, watch patterns shift, infer the rules. Backpropagation does not work that way. Without explicit explanation, the gradient update rule looks arbitrary. Creative coding education has long favored discovery-based approaches (Resnick et al. 2009; Papert 1980), while machine learning pedagogy leans on worked examples and formal exposition (Sweller et al. 1998). A curriculum that spans both domains needs a principled basis for choosing between those formats, and no published framework offers one for this particular intersection.

Then there is assessment. A standard programming test checks whether code produces the expected output. Assessment rubrics can be designed to evaluate the quality of creative products (Brookhart 2013). Generative AI education sits at the overlap: a learner must both implement an algorithm correctly and apply it with creative intent. The published record is equally thin. Grover et al. (2013) reviewed computational thinking instruments and found every one of them anchored to introductory constructs: loops, conditionals, sequences. Not a single instrument in that survey measures understanding of convolution, weight updates, or noise schedules. Tools that capture technical understanding and creative transfer in one evaluation do not, as far as the published record shows, exist.

Scaffolding, cognitive load, framework selection, assessment: the four challenges are not exhaustive, but they sit at the boundary where published knowledge runs out. The five research questions in Section 1.4 map onto them.

1.3. Research Objectives

The problem outlined above called for a response on two fronts: build the curriculum that does not yet exist, and test whether it holds up under real conditions. Six aims gave the project its shape, grouped here by purpose rather than chronological order.

On the design side, the project developed three pedagogical frameworks (Hands-On Discovery, Conceptual Deep-Dive, and Project-Based Synthesis) and distributed them across fifteen modules spanning pixel operations, cellular automata, convolution, fractal geometry, neural networks, and diffusion models. The finished curriculum was deployed as an open-access HTML site created with the Sphinx code documentation system and hosted through GitHub Pages, so that participants and future users could work through the material in a browser (Figure 1.2). Both aims fall within the "solution development" phase of the Design-Based Research cycle (Design-Based Research Collective 2003).

https://burakkagann.github.io/Pixels2GenAI/

Figure 1.2.: The full fifteen-module curriculum is deployed as an open-access website. The QR code links to the landing page (accessed March 2026).

Evaluation occupied the other half. Six of the fifteen modules were selected for a single-day workshop with nine participants whose programming backgrounds ranged from under one year to over a decade. A 24-item multiple-choice knowledge test administered before and after the session captured learning gains. The NASA Task Load Index (Hart et al. 1988) collected after every module tracked cognitive load, and exit tickets with four open-ended prompts plus one Likert item supplied qualitative and formative data. Because the two primary frameworks each governed three workshop modules, the design also permitted a matched comparison of Hands-On Discovery against Conceptual Deep-Dive on workload, self-reported understanding, and open-ended reflections. The convergent parallel mixed-methods structure follows Creswell et al. (2017). Six aims, one cycle. The final aim was to synthesize those findings into preliminary design principles for the next iteration.

The original proposal listed ten objectives, several of which fell outside this cycle: expert heuristic reviews, think-aloud protocols, self-efficacy surveys, portfolio rubrics, and TouchDesigner module testing were planned but not carried out. Section 1.5 specifies those boundaries.

1.4. Research Questions

Five research questions structured the workshop, the instruments, and the analysis. Each appears below with a note on the strength of evidence available. Their theoretical grounding is developed in Chapter 2.

RQ1 (Framework Design). What pedagogical principles and design patterns effectively scaffold learning progressions from basic array manipulation to generative AI in creative contexts?

This question received the strongest empirical attention. Two sub-questions drove the work: which instructional framework (Hands-On Discovery or Conceptual Deep-Dive) produces better outcomes for which content types, and how does module sequencing shape cumulative knowledge construction? Pre-post knowledge gains reached statistical significance with a large effect size. The matched HOD-versus-CDD comparison produced clear workload and understanding differences. Scaffolding theory supplies the conceptual backdrop (Vygotsky 1978; Wood et al. 1976).

RQ2 (Cognitive Load). How can complex technical concepts be decomposed and sequenced to maintain optimal cognitive load while building toward advanced applications?

Prior programming experience turned out to be the single strongest predictor in the dataset, correlating very strongly with average perceived workload. Two sub-questions follow: does load vary systematically across the six-module sequence, and do the modules where load peaks correspond to identifiable design choices? Cognitive Load Theory predicts that schemas built in earlier modules should absorb element interactivity in later ones (Sweller 1988); the NASA-TLX data collected after every module put that prediction to a direct test.

Two further questions carried less empirical weight. **RQ3 (TouchDesigner Integration)**, asking what strategies effectively integrate real-time visual programming systems with progressive AI learning, went untested: no TouchDesigner modules were included in the February workshop. The question is retained because Modules 10–13 in the full curriculum target that integration, and the design rationale draws on transfer-of-learning theory (Bransford et al. 1999); Chapter 6 addresses it through design argumentation. **RQ4 (Assessment)**, asking how learning outcomes in creative AI education can be assessed, was tested on one narrow slice only: whether a 24-item multiple-choice instrument reliably captures knowledge gains across the curriculum's conceptual range. Item analysis, internal consistency (KR-20), and convergent validity are reported in Chapter 5. How to assess creative application alongside technical proficiency was not tested. It remains open.

RQ5 (Transfer). To what extent do learners successfully transfer foundational concepts to novel creative AI contexts?

Transfer is hard to measure in a single day. The evidence here is indirect. Sections C and D of the knowledge test cover neural network and generative AI topics on which participants had limited direct instruction, so gains on those items serve as a rough proxy for near transfer. The drop in "I don't know" responses from 74% to 24% across the full test hints at a broader confidence shift, though it cannot separate genuine understanding from reduced test anxiety.

Exit ticket responses on inter-module connections supply the only qualitative transfer evidence collected.

The five questions depend on one another in practice. Cognitive load findings (RQ2) feed back into framework recommendations (RQ1). Assessment quality (RQ4) determines how much trust the knowledge-gain evidence for RQ1 and RQ5 deserves. That interdependence is not accidental; it follows from the Design-Based Research stance that curriculum, instruments, and theory form a single system under iterative refinement (Design-Based Research Collective 2003).

1.5. Scope and Delimitations

One Design-Based Research cycle sets the boundary for every claim made here (Brown 1992; Sandoval et al. 2004). What that cycle included, and what it left out, determines how far the findings can reach.

The curriculum spans fifteen modules, but only six were tested. Six modules spanning Module 1 (pixel fundamentals) through Module 12 (generative AI) in the full curriculum numbering were selected for the workshop because they cover the full range from the simplest entry point (RGB pixel arrays) to the most advanced topic in the current build (denoising diffusion probabilistic models). The remaining nine modules, including three TouchDesigner modules (Modules 10, 11, and 13), exist as designed artifacts but carry no participant data. Claims about their effectiveness rest on design rationale alone.

The workshop took place on a single day, February 12, 2026, and lasted approximately seven hours. Nine participants completed the session. That number sits at the low end of what Design-Based Research typically works with in a first cycle, where the priority is locating design flaws and sharpening questions, not estimating population parameters (Collins et al. 2004). Effect sizes are reported alongside p-values throughout Chapter 5, but neither should be read as stable estimates. A different nine participants, or the same nine on a different day, could shift the numbers.

Several instruments from the original proposal were not deployed. The Creative Problem-Solving Self-Efficacy Scale was dropped because the workshop schedule left no room for an additional survey battery. Think-aloud protocols were omitted to avoid disrupting the group dynamic in a shared workspace. Expert heuristic reviews were deferred to the second DBR cycle. Portfolio rubrics were not used because the workshop included no capstone project. The data that exist (knowledge test, NASA-TLX, exit tickets) cover learning gains, cognitive load, and self-reported understanding. They do not cover self-efficacy, creative output quality, or long-term retention.

The researcher served as the primary facilitator during the workshop, with the thesis supervisor present as co-facilitator to assist with logistics and technical troubleshooting. The researcher's dual role as curriculum designer and facilitator is a potential source of bias:

participants may have responded differently to exit ticket prompts knowing the person who built the curriculum was in the room. These concerns are taken up more fully in Chapter 6.

RQ3 (TouchDesigner integration) is addressed through design documentation and theoretical argument, with no empirical backing. RQ5 (transfer) rests on indirect evidence only: knowledge test section gains and the reduction in "I don't know" responses. Both questions shaped the curriculum architecture and are retained for that reason, but the reader should weigh the evidence behind them accordingly.

What follows is a first test, not a validation. Six modules were evaluated under controlled pilot conditions to extract design principles for the next iteration. The distance between a promising pilot and a validated curriculum is considerable, and nothing here should be mistaken for the latter.

1.6. Thesis Structure

Cognitive Load Theory, visual-first pedagogy, constructionism, AI literacy frameworks, and Design-Based Research each contributed a design principle to the curriculum. Where each strand falls short of the problem stated above is the focus of Chapter 2.

The research approach, workshop logistics, participant recruitment, instruments, and analytic methods are detailed in Chapter 3, which also justifies the use of non-parametric statistics at $n = 9$. The curriculum itself appears in Chapter 4 as a designed artifact: fifteen modules, three pedagogical frameworks, the technology stack, and the six workshop modules, documented so that another educator could reproduce or adapt them.

Pre-post knowledge gains, cognitive load trajectories, framework comparisons, and qualitative themes from exit tickets are reported in Chapter 5 without interpretation. All statistics follow APA conventions. Interpretation comes in Chapter 6, which reads those findings through the five research questions, revisits the literature, and addresses limitations. Chapter 7 summarizes the contributions and outlines future work.

2. Literature Review and Theoretical Framework

Five research questions drive this study, and no single theoretical lens answers all of them. Cognitive load theory, visual pedagogy, constructionism, AI literacy research, assessment methodology, and design-based research each supply part of the explanatory machinery. The sections that follow build the framework piece by piece, surfacing the gaps that the present curriculum was designed to fill.

2.1. Cognitive Load Theory in AI Education

2.1.1. CLT Foundations

Two or three interacting elements. That is roughly the limit of what working memory can process at once (Sweller et al. 1998). Cognitive Load Theory (CLT) builds on this constraint by sorting the demands placed on working memory into three categories (Sweller et al. 1998). Material that is inherently complex (many interacting elements that cannot be understood in isolation) generates intrinsic load. Instructional choices that force the learner to spend effort on tasks unrelated to the learning goal (cluttered layouts, irrelevant examples, poorly sequenced content) pile on extraneous load. Germane load is different in kind: it represents the productive cognitive work of constructing and automating schemas. When extraneous load drops, learners can invest the recovered capacity in that schema-building work (Sweller et al. 1998).

Two decades later, Sweller et al. (2019) anchored CLT in evolutionary psychology. Their updated framework separates biologically primary knowledge, skills acquired through social interaction such as spoken language, from biologically secondary knowledge, culturally transmitted content such as reading, mathematics, or programming (Sweller et al. 2019; Sweller 2024). Secondary knowledge lacks innate acquisition mechanisms, so learners depend on explicit instruction. The concept of element interactivity ties the two threads together: the more elements a task forces the learner to coordinate at once, the higher its intrinsic load. Programming, as the next paragraphs argue, is a domain where element interactivity runs especially high (Garner 2002). The capacity ceiling is real and non-negotiable.

2.1.2. CLT in Programming Education

Programming has a high intrinsic cognitive load because its elements (syntax rules, variable states, control flow, output logic) interact heavily and must be coordinated simultaneously in working memory (Garner 2002). None of these elements stands alone; each constrains the others, making programming a textbook case of high element interactivity. Garner (2002) argued for two countermeasures: worked examples that walk learners through complete solutions, and part-completion tasks that supply a partial program for the learner to finish. Worked examples reduce extraneous load by eliminating the need for means-ends problem solving; part-completion tasks then channel germane load by requiring learners to abstract the underlying schemas (Garner 2002; Salleh et al. 2018).

Empirical work backs this up. Stachel et al. (2013) built a scaffolding tool for introductory programming courses and reported trends toward lower self-reported cognitive load and higher course scores when comparing the scaffolded cohort to an unscaffolded cohort, though within-group differences were not statistically significant. Çakıroğlu et al. (2018) measured perceived load across four progressive Scratch programming concepts and found that, while the visual interface helped overall, specific graphical elements introduced unexpected extraneous load. Visual environments, in other words, are not automatically low-load. Their design must be deliberate. The Hands-On Discovery (HOD) framework in the present curriculum addresses this point directly (see Chapter 4). Deliberate design is the operative word.

2.1.3. Scaffolding as Cognitive Load Reduction

Wood et al. (1976) introduced the scaffolding metaphor to describe how a more knowledgeable partner adjusts support so that the learner operates just beyond current ability, a mechanism rooted in Vygotsky's zone of proximal development (Vygotsky 1978). Within CLT, scaffolding takes on a sharper meaning: it constrains the problem space, which lowers extraneous load and leaves more room for schema construction (Nooijen et al. 2024).

Mayer et al. (2003) proposed nine load-reduction strategies: off-loading, segmenting, pre-training, weeding, signalling, aligning, eliminating redundancy, synchronising, and individualising, each keyed to one of five overload scenarios involving combinations of essential processing, incidental processing, and representational holding (see also Mayer (2014)). The present curriculum puts several of these strategies to work, though not through a one-to-one mapping. Its six tested modules break the content into segments. Section A modules pre-train foundational array concepts before Section B introduces convolution. Colour-coded visual outputs serve a signalling function. The paired code-plus-visual format taps the off-loading principle by splitting information across verbal and pictorial channels.

Nooijen et al. (2024) reviewed 18 studies through a CLT lens and distilled expert scaffolding into two core mechanisms: cueing (directing attention) and chunking (grouping information into schemas integrated with prior knowledge). The present curriculum employs both. Colour-

coded outputs cue attention to relevant array regions; each module builds on the previous module's concepts, supplying the prior-knowledge anchors that chunking requires. In a separate discipline, Appiah-Twumasi (2024) tested scaffolding as a load reduction strategy in physics and reported significant gains ($p = .03$, $\eta^2 = .79$). The knowledge gains recorded in the present study ($W = 0.0$, $p = .008$, $d = 1.615$) similarly point to a large scaffolding effect. Salleh et al. (2018) proposed a CLT-grounded model for novice programming that pairs worked examples with goal-free problems; the Hands-On Discovery (HOD) framework operationalises a similar pairing by giving learners pre-built code patterns to extend toward a visual goal.

2.1.4. The Expertise Reversal Effect

Not all scaffolding helps all learners. Sweller (2024) described the expertise reversal effect, corroborated by a large meta-analysis (Tetzlaff et al. 2025): what supports a novice can hinder an expert, because the scaffold itself turns into extraneous load once the learner already holds an automated schema for the procedure. The worked example that orients a beginner forces the expert to process redundant information.

Tetzlaff et al. (2025) put hard numbers on this reversal. Their meta-analysis of 60 studies ($N = 5{,}924$) showed that novices gained from high instructional assistance ($d = 0.505$), while experts performed better under low assistance ($d = -0.428$). The asymmetry is substantial. Endres et al. (2023) pushed the argument further: higher prior knowledge can sometimes raise intrinsic load, because experts recognise complexities that novices miss entirely. Neurophysiological data tell a similar story. Hafezi Fard et al. (2024) used EEG data modelled through spiking neural networks to show that students with prior programming knowledge displayed reduced theta and increased alpha activity in the left frontal lobe during programming comprehension tasks, patterns associated with lower cognitive load. The workshop sample in this study ranged from less than one year to more than five years of programming experience. The correlation between prior experience and perceived cognitive load was $r_s = -.857$, $p = .007$. That value sits squarely within the expertise reversal literature and carries a practical consequence: a single, fixed level of scaffolding cannot serve all learners equally well. Future iterations of the curriculum should explore adaptive scaffolding that adjusts support to the learner's prior knowledge (see Chapter 6).

2.1.5. CLT and AI-Enhanced Learning

Recent work has begun to connect CLT with AI-based educational tools, though the empirical base remains thin. Gkintoni et al. (2025) surveyed 103 studies and argued that machine learning tools can personalise instruction in ways that manage cognitive load automatically. Qamar et al. (2025) collected survey data from 280 students and found a negative association between AI-based tool use (intelligent tutoring systems, adaptive sequencing, automated feedback) and cognitive overload. The most directly relevant study in this cluster is Koc-

Januchta et al. (2022): a two-month repeated-measures study with an AI-enriched biology textbook in which germane cognitive load was significantly higher than intrinsic or extraneous load. That pattern suggests the AI features promoted deep processing rather than surface engagement. Twabu (2025) offered a conceptual framework that integrates CLT, multimedia learning, and AI-based adaptation for distance education, though without empirical testing. The empirical base, in short, is still thin.

This study sits at a different angle within the same space. Where the studies above use AI to manage load, the present curriculum teaches AI concepts while managing load through deliberate instructional design, consistent with the explicit-instruction argument of Kirschner et al. (2006). The subject matter (generative AI) and the pedagogical strategy (progressive, scaffolded, visual-first) both engage CLT, one as content, the other as method.

2.1.6. Visualization as Cognitive Load Reduction

Visualisation connects CLT to the broader pedagogical approach discussed in Section 2.2. Winter et al. (2019) applied segmentation and subgoal labelling through a visual programming environment to teach threshold concepts in programming and found improved mastery of problem decomposition in the treatment group. The logic is simple. When part of the representational work moves from working memory to an external display, extraneous load drops (Mayer 2014). The Hands-On Discovery (HOD) modules in the present curriculum follow this principle: each module pairs code with immediate visual output (pixel arrays, fractal patterns, cellular automata), converting an internal cognitive process into an inspectable artifact. The NASA-TLX data support the interpretation. Hands-On Discovery (HOD) modules recorded lower workload scores than Conceptual Deep-Dive (CDD) modules ($p = .016$, $d = 1.10$).

What visualisation contributes beyond load reduction, through dual-channel encoding and spatial cognition, is the question that follows.

2.2. Visual-First Pedagogy and Dual Coding Theory

2.2.1. Theoretical Foundations: Multimedia Learning

Humans drew before they wrote. Tversky (2011) traced this priority through cognitive science, arguing that spatial representations abstract and schematise information in ways that complement, and often precede, verbal encoding. Depictive expressions such as maps, diagrams, and gestures exploit the spatial properties of thought itself; they are not decorative supplements to language but a parallel channel for reasoning. Paivio (1986) formalised a version of this claim as dual coding theory: cognition operates through two systems, one processing verbal material and the other processing images, and information encoded through both is retained more reliably than information processed through either alone.

Three assumptions anchor the cognitive theory of multimedia learning (CTML) advanced by Mayer (2014). Learners possess separate channels for verbal and pictorial material; each channel has limited capacity; and meaningful learning requires active processing across both (Mayer et al. 2003; Mayer 2014). The practical consequence is that words paired with pictures produce deeper learning than words on their own. CTML gives the present curriculum its structural logic. Every module pairs code with an immediate visual output, whether a pixel array, a fractal, a cellular automaton, or a diffusion sequence, so that both channels carry instructional weight on every task. Two channels carry more than one.

2.2.2. Visualization in Programming Education

Syntax, variable bindings, and control flow are all symbolic, and a novice programmer must juggle these interacting symbols in working memory at the same time (Garner 2002). That burden falls almost entirely on the verbal channel (Mayer 2014). Visualization can redistribute part of it. Baldwin et al. (2000) sorted programming knowledge into three types, syntactic, conceptual, and strategic, and argued that visual interfaces can support each type. Colour-coded highlighting, for instance, aids syntactic recall; diagrams map data structures for conceptual grasp; animated traces make algorithmic choices inspectable, which builds strategic competence. Kelleher et al. (2005) surveyed visual programming environments broadly and reached a similar conclusion: lowering the notational barrier through graphical representations widens the entry point for novice learners.

Chung (2025) put these ideas into practice with a web-based platform that renders code output in real time alongside the source; usability testing with beginners showed sustained engagement on tasks that typically provoke early dropout. Gonzalez et al. (2024) inverted the direction: instead of providing visualisations, the researchers asked novices to draw their own representations of code constructs. The drawings varied enormously. That variation exposed misconceptions that verbal explanations alone would have missed. On the feedback side, Marwan et al. (2022) showed that adaptive immediate feedback in block-based programming raised both completion rates and performance scores. The common thread is simple. Visual feedback, whether system-generated or learner-produced, tightens the loop between action and understanding.

2.2.3. Visualizing Neural Networks and AI Concepts

Visualisation arguably matters even more for deep learning than for introductory programming, because the objects of study (weight matrices, feature maps, loss surfaces) have no natural physical form. Several tools have tested whether interactive graphics can fill that gap. Wang et al. (2021) built CNN Explainer, a browser-based tool that links an architectural overview of a convolutional neural network to elastic detail views and live formula rendering. Non-expert users could trace how a single input pixel propagated through convolutional, pooling, and fully connected layers without reading the underlying equations in isolation. CNN 101, a

companion project, coupled high-level diagrams with low-level mathematical animations so that learners could zoom between structure and arithmetic at will (Wang et al. 2020b). Huang et al. (2023) pushed the multi-level idea further with ConceptExplainer, a tool that lets users inspect deep neural networks at global, class, and instance levels through concept-based summaries rather than raw activations.

At a smaller scale, Bäuerle et al. (2022) ran a between-subjects experiment ($n = 37$) comparing exploRNN, an interactive recurrent neural network visualisation, against a static alternative. Participants in the interactive condition reported significantly less extraneous cognitive load, a result that ties visual interactivity directly to the CLT framework in Section 2.1.

2.2.4. Interactivity as a Necessary Condition

Not every visual helps. Kirchler et al. (2021) tested whether static explanation methods such as saliency heat maps genuinely improved user understanding of a classifier's decisions. Participants who viewed the heat maps said they understood the classifier, but their actual comprehension, measured through prediction tasks, was no better than chance. Only when the researchers introduced an interactive tool built around a generative model (StyleGAN2) could users probe the classifier's reasoning and uncover biases that static displays had hidden entirely. The warning is direct: a curriculum that shows learners a neural network diagram and moves on risks producing false comprehension. Live code execution with manipulable parameters, the approach the present study adopts, is harder to build but harder to fake understanding with. Adhikari (2023) arrived at a compatible conclusion from a different angle, proposing that process visualisations, visual records of how a learner reached an output rather than the output itself, supply metacognitive feedback that static artifacts cannot match.

2.2.5. From Concrete to Abstract: Visual Pathways to AI

Queiroz et al. (2020) made explicit what the tools above leave implicit: learners should encounter AI through concrete, manipulable representations before confronting abstract formalisms. Their AIcon2abs methodology, echoing Bruner's concrete-to-abstract sequence (Bruner 1960), teaches artificial intelligence to the general public, including children, through visual and block-based programming paired with weightless neural networks. Training and classification in AIcon2abs are not hidden behind library calls. They are programming constructs that learners assemble, run, and watch. The trajectory moves from physical classroom exercises through block-based visual programs to text-based implementations, a concrete-to-abstract arc that mirrors the architecture of the present curriculum. Where AIcon2abs targets weightless neural networks, the "Arrays to Art" curriculum substitutes a longer progression, from NumPy arrays through convolution and perceptrons to denoising diffusion, but the underlying pedagogical logic is the same: ground every abstraction in something the learner can see and change.

Seeing a result, however, is not the same as constructing the artifact that produces it. Constructionist theory, examined in the next section, reframes the learner's role from observer to builder and explains why that shift matters for the type of deep schema construction that both CTML and CLT treat as germane processing (Section 2.3).

2.3. Constructionism and Creative Computing

2.3.1. Constructionist Foundations

Papert (1980) gave children a programming language called Logo and asked them to command a screen turtle to draw geometric shapes. The point was not the language. It was the tight coupling between instruction and visible consequence: type a command, watch the turtle move, revise. Harel et al. (1991) later named the underlying principle constructionism, setting it apart from Piaget's constructivism by placing the public, shareable artifact at the centre of the learning process rather than treating it as an incidental product. On this view, the artifact is where learning happens, not what is left over after learning has occurred.

Vygotsky (1978) provides a developmental mechanism for why this works. Guided support lets a learner accomplish tasks just beyond current ability; constructionist environments embed that guidance in the artifact itself. A pixel array that turns red when the code is wrong, a fractal that collapses when the recursion depth is off, a cellular automaton that freezes when the rule table has a gap: each failure state delivers feedback that is situated, immediate, and specific. No tutor needs to intervene. Brennan (2015) drew a useful line here, arguing that constructionist teaching means learning *with* and *through* technology rather than merely learning *about* it (Brennan 2015, p. 289).

2.3.2. Creative Coding as Pedagogy

Scratch made constructionism scalable. Resnick et al. (2009) designed the platform so that young people could be producers, not just consumers, of digital media, and millions of shared projects proved that an environment with what Resnick called a "low floor, wide walls, and high ceiling" (Resnick et al. 2009, p. 62) could sustain both beginners and advanced users. Resnick (2007) extended the argument beyond childhood, proposing that iterative cycles of imagining, creating, playing, sharing, and reflecting are productive well beyond childhood. Processing, the Java-based creative coding framework, carried a similar philosophy into higher education and the arts (Processing Foundation 2024). Peppler et al. (2005) tracked how media-arts workshops using these tools turned youth into active producers of digital culture; computational skills came along almost as a side effect.

The approach holds up across populations. A multi-year, multi-site implementation of a Processing-based creative computation curriculum worked for both high school and college students regardless of prior background (Xu et al. 2018). Terroso et al. (2022) taught program-

ming to non-CS master's students through p5.js and reported that the visual, project-based format sustained motivation in a cohort that had previously resisted text-based instruction. In architecture education, Domínguez-Gómez et al. (2024) found strong parallels between architectural problem-solving and computational thinking, evidence that creative coding carries across disciplinary lines. The most controlled test came from Soh et al. (2015): computational creativity exercises produced significantly higher CT scores than standard exercises ($F(1, 106) = 12.78$, $p < .01$), and the effect grew with dosage. More creative exercises meant larger gains.

2.3.3. Scaffolding Within Creative Environments

Creative tools alone do not guarantee learning. Johnson (2017) found that Game Maker helped some Year 9 pupils learn programming but left others behind; the open-ended environment offered too little structure for novices who lacked the schemas to navigate it. In science education, Hoban et al. (2010) saw their "5 Rs" multimodal framework succeed only when students moved through a structured sequence of representational activities rather than jumping straight into animation production. Even at-risk youth in juvenile detention could engage with constructionist projects, but only when facilitators supplied sustained, individualised scaffolding (Stager 2005). The pattern is consistent: openness without structure produces uneven results.

Transfer is harder still. Lobo et al. (2025) studied a balanced creative coding curriculum and found that students picked up fundamentals but could not carry complex data structures into independent projects. That gap between guided construction and independent application recurs in the present study, where Section D (generative AI) post-test gains were the weakest across all content sections. Wu et al. (2023) tackled a different facet of the problem with a Scratch-based tracking system that let teachers spot weaknesses in real time; the exit ticket system in the present curriculum plays a similar diagnostic role, catching misconceptions module by module rather than at the end.

2.3.4. Gap: Constructionism and Generative AI

No study in the literature reviewed here extends the constructionist arc from foundational computing through to generative AI. The present curriculum attempts exactly this. Learners construct visual artifacts at every stage: RGB pixel arrays in Module 1, cellular automata and fractals in Modules 2 and 4, perceptron decision boundaries in Module 5, denoising diffusion sequences in Module 6. The "art" in "Arrays to Art" is not ornamental. It is the constructionist artifact through which each technical concept becomes inspectable, shareable, and revisable. How learners at different experience levels engage with this progression is a question that the AI literacy frameworks in the next section begin to address (Section 2.4).

2.4. AI Literacy Frameworks

2.4.1. Defining AI Literacy

Long et al. (2020) organised AI literacy around five thematic areas: recognising AI, understanding its representations, perceiving intelligence, deciding how to interact with it, and reasoning about its societal implications. The framework drew on education, psychology, and HCI scholarship and was designed to apply across age groups and disciplinary boundaries. Ng et al. (2021) refined the construct along four dimensions, know and understand AI, use and apply AI, evaluate and create AI, and engage ethically with AI, arguing that literacy must extend beyond conceptual knowledge into skilled, critical practice. The two accounts overlap substantially, but Ng and colleagues placed greater emphasis on ethical reasoning as a standalone dimension rather than embedding it within general societal awareness.

Several recent frameworks have translated these broad competencies into staged curricula. MacCallum et al. (2024) introduced the Scaffolded AI Literacy (SAIL) model, which arranges learner progression across four levels: Know, Use, Evaluate, and Beyond. Each level assumes mastery of the previous one, creating a dependency chain that mirrors CLT-informed sequencing (Section 2.1). Mills et al. (2024) arrived at a comparable structure for K–12 contexts, organising AI literacy around three modes of engagement: Understanding, Evaluating, and Using AI. The pattern holds in higher education too. Where MacCallum and Mills sequence competencies linearly, Zhou et al. (2024) took a different route and aligned their framework explicitly with Bloom's revised taxonomy, mapping AI competencies onto remembering, applying, analysing, and creating. The three frameworks differ in granularity and target population, yet they converge on one claim: AI literacy is not a single threshold that learners cross. It is a progression, and curricula should scaffold that progression deliberately. The present curriculum operationalises this principle. Modules 1–4 map to SAIL's Know and Use levels, Modules 5–8 to Use and Apply, Modules 9–12 to Evaluate and Create, and Modules 13–15 to Beyond (see Chapter 4). Literacy is a staircase, not a doorway.

2.4.2. Teaching AI/ML to Non-Experts: Barriers and Approaches

Scaffolding is easy to prescribe and hard to execute, particularly when learners carry misconceptions about how machines process information. Sulmont et al. (2019) catalogued pedagogical content knowledge (PCK) for machine learning instruction and identified several barriers that block novice understanding, among them conflating human reasoning with computational processes, expecting deterministic outputs from probabilistic systems, and attributing intentionality to algorithmic behaviour. Related instructional challenges include the difficulty of making model internals visible and the absence of concrete analogies for gradient-based learning. Yang et al. (2018) documented similar pitfalls in a study of non-experts building ML classifiers: participants struggled to select meaningful features, misinterpreted model confidence, and rarely tested edge cases. Their proposed remedy, "Test-Driven Ma-

chine Teaching," asks learners to specify expected outputs before training, which forces explicit reasoning about what the model should learn. Both studies locate the core problem in the gap between what learners assume (human-like reasoning) and what the system actually does (numerical optimisation over arrays).

Successful interventions have bridged that gap through hands-on, visually grounded activities. Carney et al. (2020) reported that Google's Teachable Machine generated over 125,000 user-built models across 201 countries within its first year; the tool lets users train image, sound, and pose classifiers through a drag-and-drop browser interface, with no code required. Sydora et al. (2026) embedded k-nearest-neighbour and linear regression concepts inside LEGO robotics workshops for participants aged 12–17. Learners programmed physical robots to classify sensor data, which made the mapping from input to prediction tangible. Chen et al. (2024) tested MindScratch, a Scratch-based environment enhanced with multimodal generative AI support, and found that fifth-graders in the treatment group scored higher on both computational thinking and creative output measures. A common thread runs through all three: each intervention pairs direct manipulation with immediate, visual feedback. That pairing echoes both the multimedia principles discussed in Section 2.2 and the load-reduction logic of Section 2.1. In the present study, Modules 5 and 6 (Perceptron and DDPM) recorded the highest NASA-TLX scores across all six tested modules, a pattern consistent with the barriers that Sulmont et al. (2019) and Yang et al. (2018) described; the strong inverse correlation between prior programming experience and perceived load ($r_s = -.857$, $p = .007$) reinforces the point that prior knowledge matters when abstract AI concepts enter the picture. Prior knowledge is not optional baggage.

2.4.3. Gap: Progressive Technical Pathways to Generative AI

The tools surveyed above share a limitation. Teachable Machine, MindScratch, and the LEGO robotics platform all target classification or recognition tasks. Learners train a model to sort inputs into categories; they do not build or inspect systems that generate new outputs. Suh et al. (2022) moved closer to generative territory with CodeToon, which uses conversational AI to scaffold computational thinking through comic creation, and Wang (2025) explored how generative AI tools can support creative production in business education. Neither, however, asks learners to understand the technical machinery behind generation itself. No published curriculum, to the knowledge of the author, traces a progressive path from basic array manipulation through convolutional operations and single-neuron models to the architecture of denoising diffusion probabilistic models. That gap defines the contribution of the present study. The workshop evaluation of Modules 1–6 produced significant knowledge gains ($W = 0.0$, $p = .008$, $d = 1.615$), suggesting that the first half of this progression is viable. Whether the remaining modules sustain those gains is a question for future iterations.

2.5. Assessment in Creative-Technical Domains

2.5.1. Assessing Computational Thinking and Programming Knowledge

Brennan et al. (2012) proposed a three-dimensional framework for computational thinking (CT) assessment that separates concepts (sequences, loops, conditionals), practices (debugging, testing, reusing), and perspectives (expressing, connecting, questioning). That framework shaped how CT instruments were built over the next decade, yet the instruments that followed vary wildly in rigour. Corrales-Álvarez et al. (2025) reviewed 17 university-level CT instruments and found psychometric validation sparse; most tools offered only face validity or basic reliability coefficients. Ocampo et al. (2024) reached a harsher verdict across a broader sample: only 54% of 50 surveyed instruments presented any formal validity evidence, leaving nearly half without it. The field, in short, builds tests faster than it validates them.

A handful of studies buck that trend. Guggemos et al. (2019) applied both item response theory and confirmatory factor analysis to a CT test administered in Swiss upper-secondary schools, producing item-level fit statistics rarely seen in the literature. Rohaeti et al. (2025) used Rasch modelling to calibrate a computational thinking instrument for Indonesian pre-service teachers, with person and item reliability above .80. Aristizábal Zapata et al. (2024) combined Kuder–Richardson 20 (KR-20) with Aiken's V coefficient to validate a 40-item computational thinking test for children, achieving an overall KR-20 of .838. These exemplars share a commitment to going beyond Cronbach's alpha, a step the present study also takes. The 24-item knowledge test used in this workshop maps to four content sections: A (array fundamentals), B (visual computing), C (neural networks), and D (generative AI). Its post-test KR-20 of 0.913 meets or exceeds all benchmarks reported in the studies above (see Chapter 5).

Luxton-Reilly et al. (2018) argued that conventional programming assessments fail because they bundle too many concepts into single items, a concern echoed in broader CT assessment reviews (Grover et al. 2013). A question that tests loop syntax, variable scope, and output formatting simultaneously cannot tell the instructor which concept the student missed. Decomposing assessments into atomic elements, each targeting one concept, enables accurate diagnosis. The section-mapped structure of the present instrument (six items per content section, each section targeting a distinct knowledge domain) operationalises that decomposition principle. One concept per item; no hiding places.

2.5.2. Creative Coding Rubrics and Portfolio Assessment

Assessing knowledge is one thing. Assessing creative output alongside technical skill is harder, and the literature has not settled on how to do it. Blake-West et al. (2024) developed a Creative Coding Rubric, tested it with 1,201 student projects, and found an internal consistency of $\alpha = .93$. Their factor analysis, however, turned up a surprising result: design quality and coding quality loaded onto a single dimension rather than two. If

that holds, the finding challenges the assumption that creative and technical competence are separable constructs. Basu (2019) arrived at a compatible conclusion from a different direction, arguing that rubrics separating "design" from "code" misrepresent how students actually work, because the two activities interleave continuously during creative coding.

Portfolio assessment offers an alternative path. Cassidy (2008) documented a feasible portfolio-based approach to computing assessment but flagged the workload cost: grading portfolios took roughly three times longer than grading tests. Scaling remains an open problem. Alves et al. (2021) surveyed the field and found no working consensus on how to define, let alone measure, creativity in computing. Putra et al. (2022) pointed to automated creativity scoring as a possible way forward, though their prototype handled only block-based programs.

The present study chose objective knowledge testing over creative assessment. That was a deliberate scope decision, not an oversight. Given the field's immaturity on creative measurement, a validated knowledge test with strong reliability (KR-20 = 0.913) provides a firmer evidentiary foundation than an unvalidated rubric would have.

2.5.3. Methodological Considerations: Pre-Post Design and Transfer

Pre-post designs remain the workhorse of short-format educational evaluation, but their limits are well documented. Gouldthorpe et al. (2013) recommended them for factual knowledge gains while cautioning that attitudinal or perceptual change is better captured through retrospective or mixed-method designs. Jayaratne et al. (2025) tested that boundary empirically, finding that combining objective measures with subjective self-reports and retrospective items produced the most complete picture of workshop impact. The present study follows that logic: the knowledge test supplies objective data, NASA-TLX captures subjective load, and exit tickets collect retrospective reflections module by module. Convergent validity strengthens the case. Román-González et al. (2019) showed that scores on three independent CT instruments correlated positively and significantly, building on Román-González et al. (2017), suggesting they tap the same underlying construct. A parallel pattern appears in the present data, where self-rated confidence on exit-ticket question 4 correlated negatively with NASA-TLX composite scores ($r_s = -.514$, $p < .001$), which supports the interpretation that the two instruments tap related but inverse facets of the learning experience.

Jordan et al. (2018) evaluated Software Carpentry and Data Carpentry workshops, short-format, pre/post, skills-focused, and found significant self-reported gains across programming, data management, and confidence. That study is the closest published design parallel to the present one, with one key difference: the Carpentries evaluation relied entirely on self-report, while this study adds an objective 24-item test and includes a "do not know" (IDK) option that tracks metacognitive change. The IDK reduction from 74% to 24% is a measure no standard pre-post design captures. Hölbl et al. (2019) found that students who rated their own competence lower before instruction tended to show larger measured gains; the present study found a non-significant positive correlation ($r_s = +.204$) between prior experience and

knowledge gains, suggesting that in this sample, everyone learned regardless of starting point.

Transfer is the hardest outcome to demonstrate. Scherer (2016) conducted a critical review of programming transfer studies and found the evidence base broadly weak, with effect sizes small and inconsistent across populations. Ezeamuzie (2022) reached a similarly inconclusive verdict for CT transfer. Against that backdrop, the Section D results in the present study (Hedges' $g = 0.271$, 56% residual IDK responses) look less like a failure and more like a field-wide pattern. Transfer from guided instruction to genuinely novel domains remains an open challenge in computing education, not a shortcoming specific to this curriculum.

2.6. Design-Based Research Methodology

2.6.1. Origins and Principles of DBR

Laboratory findings about learning rarely survived contact with the noise, interruptions, and social dynamics of real classrooms. Brown responded with what she called "design experiments" (Brown 1992, p. 141): research conducted inside those classrooms rather than apart from them. Classrooms are not broken versions of labs. They are the actual site where instruction succeeds or fails, and Brown argued that researchers should treat that messiness as informative data, not as a confound to eliminate. Collins et al. (2004) built on this stance, recasting design research as a methodology with explicit dual goals: refine the intervention under study while simultaneously advancing theoretical understanding of the learning processes it triggers. A curriculum revision that raises test scores but teaches the researcher nothing about why those scores moved is, on this account, half a success at best.

Five characteristics set DBR apart from conventional experimentation, according to the Design-Based Research Collective (2003). Design activity and research inquiry develop in lockstep, so that the artifact and the knowledge about learning co-evolve. Development moves through repeated cycles: build, enact, analyse, rebuild. The process must generate theories concrete enough for other practitioners to adapt. All of this takes place in authentic settings, not sanitised surrogates. And any account of the design must explain how it functions under those real conditions, not merely report that it did. Reeves (2006) compressed these principles into a four-phase workflow: identify the practical problem, develop a theoretically grounded solution, run iterative rounds of testing and refinement, and distil transferable design principles from the accumulated evidence. The model gives practitioners a concrete path without collapsing the methodology into a recipe.

One distinction matters for what follows. DBR does not try to "prove" that an intervention works the way a randomised controlled trial would. Its aim is to explain which features of a design produce which learning outcomes, and in what contexts those features hold or break down. That orientation fits the present study precisely. Showing that the "Arrays to Art" curriculum raises test scores is necessary but not sufficient (the scores did rise; see

Chapter 5). The deeper question is which design patterns, progressive sequencing, visual-first scaffolding, alternation between instructional frameworks, carry the learning gains, and which ones add friction without payoff. That is the question this study pursues.

2.6.2. DBR in Computing and AI Education

Each cycle of design-based research produces what Cobb et al. (2003) described as relatively humble theories: local, domain-specific conjectures about how particular design features interact with learner characteristics to shape outcomes (see also Sandoval et al. (2004)). The qualifier is deliberate. No single cycle settles a question. What it does is shrink the space of plausible explanations and sharpen the conjectures that the next round puts to the test. That logic makes DBR a natural fit for educational technology (Wang et al. 2020a), where the intervention, the delivery platform, and the pedagogy routinely shift in tandem. Nothing holds still long enough for a single-variable experiment.

Several studies reviewed earlier in this chapter follow workshop-test-refine rhythms that are characteristic of DBR, though their authors do not always use the term. Sydora et al. (2026) ran LEGO robotics workshops teaching machine learning fundamentals to adolescents across two iterations (Section 2.4). Jordan et al. (2018) evaluated Software and Data Carpentry workshops with pre-post and long-term surveys, within an organisation that systematically uses such data to refine its offerings (Section 2.5). Neither study frames itself as DBR. Both instantiate its core loop.

The case for DBR grows stronger when the field itself is young. AI literacy education, as Section 2.4 argued, sits at that early stage. Curricula change between semesters, pedagogical strategies lack validation, and the target competencies are still being negotiated (Long et al. 2020; Ng et al. 2021). No stable "treatment" exists for a controlled experiment to isolate. What the field needs are structured cycles that build instructional conjectures and test them in the same motion. DBR supplies that structure.

2.6.3. This Study as First-Cycle DBR

This study is one pass through Reeves' cycle: identify a gap, build a curriculum, test it with learners, and extract design principles from the evidence. Reeves' four phases map onto the present study in a direct way. Problem analysis (Phase 1) took the form of the literature review in this chapter, which surfaced a specific gap: no published curriculum traces a progressive technical pathway from array manipulation through to generative AI (Section 2.4.3). Solution development (Phase 2) produced a 15-module curriculum built around two instructional frameworks, Hands-On Discovery (HOD) and Conceptual Deep-Dive (CDD), grounded in CLT (Section 2.1), constructionism (Section 2.3), and visual pedagogy (Section 2.2). Iterative testing (Phase 3) ran as a single-day workshop on February 12, 2026: nine participants worked through Modules 1–6, evaluated via a 24-item knowledge test, per-module NASA-TLX ratings, and qualitative exit tickets. Reflection (Phase 4) is this thesis

itself, which documents the design principles that the first cycle surfaced; Chapter 5 reports the evidence and Chapter 6 interprets it.

A single cycle has clear limits. DBR's explanatory power builds across iterations, as each round tightens the conjectures the next round tests (Cobb et al. 2003; Reeves 2006). The findings here are therefore first-cycle hypotheses, not final answers. Take the Hands-On Discovery (HOD) versus Conceptual Deep-Dive (CDD) comparison: the hypothesis that output-driven modules produce lower cognitive load than concept-driven modules received initial support ($p = .016$, $d = 1.10$), but the result rests on six modules and nine participants. That is a grounded conjecture, not a verdict. Future cycles would extend testing to Modules 7–15, bring in the TouchDesigner integration that RQ3 targets, and recruit larger and more varied samples. How Phase 3 was conducted, including sampling, instrumentation, data collection procedures, and analysis, is the subject of Chapter 3.

2.7. Gaps in Existing Knowledge

Six bodies of literature converge on the problem this curriculum addresses: cognitive load theory, visual-first pedagogy, constructionism, AI literacy frameworks, assessment in creative-technical domains, and design-based research methodology. Each strand is well developed on its own terms. Their intersection, however, exposes gaps that no single study has addressed. Five stand out, and together they mark the space the present curriculum occupies.

The AI literacy tools surveyed in Section 2.4 target classification and recognition. Teachable Machine trains image classifiers. MindScratch wraps machine learning inside Scratch blocks. LEGO robotics workshops teach k-nearest-neighbour on sensor data. None of these systems ask the learner to build or inspect a generative model. The constructionist environments reviewed in Section 2.3 face a parallel ceiling: Scratch, Processing, and p5.js support creative coding but rarely extend to neural network architectures, let alone diffusion processes. Existing curricula cover subsets of this territory: array-level image processing here, classification tasks there, block-based ML elsewhere. What none of them attempt is a single, unbroken progression that carries learners from pixel-level array work through convolution and perceptron training all the way to diffusion-based generation. That is the trajectory the present study fills.

Section 2.1 covered extensive CLT research in programming education and noted that the CLT-AI intersection is just beginning to accumulate evidence. The studies that do exist use AI to manage load; they do not teach AI concepts under controlled load conditions. No study has tracked cognitive load across a progressive AI curriculum at module-level granularity using NASA-TLX. The expertise reversal effect, well established in programming contexts ($d = 0.505$ for novices, $d = -0.428$ for experts in the Tetzlaff meta-analysis) (Tetzlaff et al. 2025), has not been tested in creative AI settings. The present study supplies the first module-by-module load trajectory for an AI literacy curriculum, including a strong experience-load correlation ($r_s = -.857$, $p = .007$) that ties directly to the reversal literature.

Computational thinking instruments are plentiful but domain-generic, as Section 2.5 showed, and more than half of the instruments surveyed in that section lack formal validity evidence. No instrument specifically targets learning progression from array manipulation through visual computing to generative AI concepts. The 24-item knowledge test built for this study (KR-20 $= 0.913$ post-test) fills that gap. Its section-level item mapping (A through D) enables fine-grained diagnosis of where learning occurs and where it stalls.

Section 2.2 reviewed interactive visualisation tools for neural network understanding (CNN Explainer, exploRNN), and Section 2.3 surveyed creative coding platforms (Processing, p5.js). These two worlds have not converged. No study has investigated whether real-time visual programming environments such as TouchDesigner can scaffold progressive AI learning. This gap maps onto RQ3. The present thesis addresses it through design documentation; empirical testing is left for future DBR cycles.

Transfer from programming instruction to broader computational thinking remains inconclusive, as Section 2.5 concluded. Transfer from foundational computing concepts to creative AI application is essentially unstudied. Section D knowledge gain in the present workshop (Hake's normalised gain $g = 0.271$, 56% residual IDK) and the overall IDK reduction from 74% to 24% offer preliminary evidence, but the base is thin. The field has little to build on here.

Table 2.1 summarises the five gaps, their evidence status from the present study, and the chapters in which each is addressed.

Table 2.1.: Summary of identified research gaps, evidence status, and thesis chapters addressing each gap

Gap	Description	Evidence Status	Addressed In
1	No progressive curriculum from arrays to generative AI	Empirical ($W = 0.0$, p = .008, d = 1.615)	Chapter 4, Chapter 5
2	No cognitive load data for progressive AI curricula	Empirical (r_s = -.857, p = .007)	Chapter 5, Chapter 6
3	No validated assessment for progressive AI literacy	Empirical (KR-20 $= 0.913$)	Chapter 3, Chapter 5
4	No real-time creative system integration with AI pedagogy	Theoretical only	Chapter 4
5	Weak transfer evidence in creative computing	Preliminary (Hake's $g = 0.271$, 56% residual IDK)	Chapter 5, Chapter 6

Note. $n = 9$, six modules tested. IDK = "I don't know" test responses.

One cycle, nine participants, six of fifteen modules: the scope is bounded, and the findings are best read as grounded conjectures rather than settled conclusions. Chapter 3 describes how the investigation was carried out.

3. Methodology

On February 12, 2026, nine participants completed a single-day workshop that tested six modules drawn from a fifteen-module curriculum. Data collection relied on five instruments: a demographics questionnaire administered before the session, a 24-item pre-post knowledge test, the NASA Task Load Index (Hart et al. 1988) completed after each module, open-ended exit tickets collected alongside each TLX administration, and a facilitator observation protocol. Of these, the knowledge test, the NASA-TLX, and the exit tickets supplied the primary quantitative and qualitative evidence. Together, the instrument set needed to address five research questions covering curriculum design (RQ1), cognitive load (RQ2), tool integration (RQ3), assessment (RQ4), and transfer (RQ5). Design-Based Research framed the methodology. Convergent parallel mixed methods (Creswell et al. 2017) organized data collection, with quantitative and qualitative strands running side by side and combined during analysis. Table 3.1 maps each instrument to its timing and data type.

Table 3.1.: Overview of data collection instruments.

Instrument	**Timing**	**Data Type**	**Primary RQ(s)**
Demographics questionnaire	Pre-session	Categorical / ordinal	Context
Knowledge test: pre (24 MC items)	Pre-session	Dichotomous (0/1)	RQ1, RQ4, RQ5
NASA-TLX ($\times 6$)	After each module	Ordinal (1–20)	RQ2
Exit tickets ($\times 6$)	After each module	Open-ended + Likert	RQ1, RQ2, RQ5
Facilitator observation protocol[a]	During session	Field notes	RQ3
Knowledge test — post (24 MC items)	Post-session	Dichotomous (0/1)	RQ1, RQ4, RQ5
Post-workshop evaluation form	Post-session	Likert + open-ended	Supplementary

[a]Notes were kept throughout the session but were not coded systematically before the submission deadline.

3.1. Research Approach: Design-Based Research

Standard experimental designs in education depend on control groups and random assignment to isolate causal effects (Creswell et al. 2017). None of these applied here. The curriculum under evaluation did not exist before this thesis. One researcher designed it and tested it within a single academic term. No comparable generative AI curriculum existed in the published literature at the time, ruling out a control condition. The research questions themselves shifted as modules took shape. These constraints match the conditions that gave rise to Design-Based Research (Section 2.6): a novel intervention, no viable comparison group, and an evolving research focus. Anderson et al. (2012) reviewed a decade of DBR

studies and concluded that the methodology is strongest when the object of study is still being built, which is precisely the situation a first-draft curriculum presents. Collins et al. (2004) made the complementary case that stripping away contextual complexity to achieve experimental control sacrifices the very conditions under which learning actually happens.

The methodology's core commitments, reviewed in Section 2.6.1, played out in three specific ways during this project:

- **Interventionist.** Fifteen modules were built around a single question: whether a progression from pixel-level array operations to generative AI models could work as a teaching sequence. The researcher designed the learning environment rather than observing an existing one.
- **Theory-driven.** Cognitive Load Theory shaped how many concepts each module introduced and how steeply the mathematical notation escalated, while workshop results were meant to feed back into revised design principles. That feedback loop (theory informing design, data refining theory) is what makes DBR theory-driven rather than purely empirical (Design-Based Research Collective 2003).
- **Iterative (limited).** Cobb et al. (2003) envisioned repeated design-test-revise cycles that converge over time on stable instructional theory. One cycle is all this study completed. Internal revisions occurred during module development, but participants encountered the curriculum exactly once.

Section 2.6.3 mapped this study onto the four-phase DBR framework of Reeves (2006). What follows is the concrete timeline. Development ran from October 2025 through early February 2026 and produced fifteen modules, drawing on the gap identified during the literature review (Chapter 2): no existing curriculum traced a continuous path from low-level array manipulation to generative models. Three pedagogical frameworks were selected as scaffolding templates, Hands-On Discovery (HOD), Conceptual Deep-Dive (CDD), and Project-Based Synthesis (PBS), each suited to a different type of content. Of the fifteen modules, six were chosen for workshop testing. The selection aimed to cover the full difficulty range of the curriculum, from pixel-level color operations through procedural generation, simulation, classical image processing, and neural network training, while representing both the HOD and CDD instructional approaches. No Project-Based Synthesis (PBS) module had reached a testable state by the evaluation date, so that framework went untested.

The workshop itself took place on February 12, 2026. Nine participants worked through modules M1 through M6 in a fixed sequence, completing a pre-test beforehand, a NASA-TLX survey and an exit ticket after each module, and a post-test at the end. The empirical findings from that day feed into the design principles discussed in Chapter 6, closing the first pass through the DBR cycle. Figure 3.1 maps these four stages onto the thesis timeline.

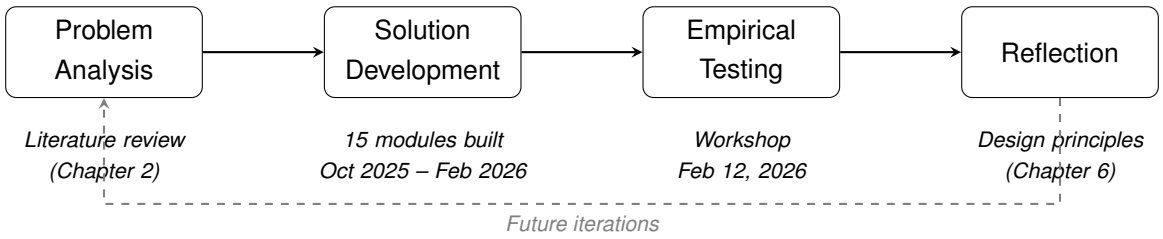


Figure 3.1.: Design-Based Research cycle after Reeves (2006), mapped onto the thesis timeline. Solid arrows trace the single cycle completed in this study. The dashed return arrow indicates the iterative loop that characterizes mature DBR but was not completed here.

A single evaluation cycle is a real constraint, as Section 2.6.3 noted. One pass through the design-test-reflect loop puts a ceiling on how far the resulting principles can generalize (Collins et al. 2004). The numbers underscore the point: $n = 9$ does not support population-level claims, and six modules out of fifteen is less than half the planned curriculum. First-cycle studies still contribute when the researcher documents every design decision and reports unfavorable results alongside favorable ones (Design-Based Research Collective 2003). Both conditions are met here. Out of that single cycle came nine participant trajectories through six modules. Pre-post knowledge data, per-module workload ratings, and qualitative reflections point in the same direction for some research questions and pull apart for others.

3.2. Study Design

Nine participants completed quantitative instruments and qualitative prompts during the same workshop session, on the same day, in the same room. The quantitative strand consisted of a 24-item pre-post knowledge test and six rounds of the NASA Task Load Index. The qualitative strand consisted of open-ended exit ticket responses collected after each module: four prompts per round, producing up to 216 text entries across all participants and modules. Creswell et al. (2017) label this arrangement *convergent parallel*: both strands were collected simultaneously, analyzed separately, and merged during interpretation.

The split exists because the two strands answer different parts of the same questions. Test scores and TLX ratings put numbers on *what changed* and *how hard it felt*. Exit ticket narratives supply the *why*. A statistically significant load difference between two modules, on its own, gives no hint about what caused it. Conversely, one participant's remark about confusing notation cannot tell us whether the confusion was shared by the group. When both strands point to the same conclusion, confidence in that conclusion goes up; when they pull apart, the mismatch itself becomes a finding worth investigating (Creswell et al. 2017). Figure 3.2 traces the two strands from collection through integration.

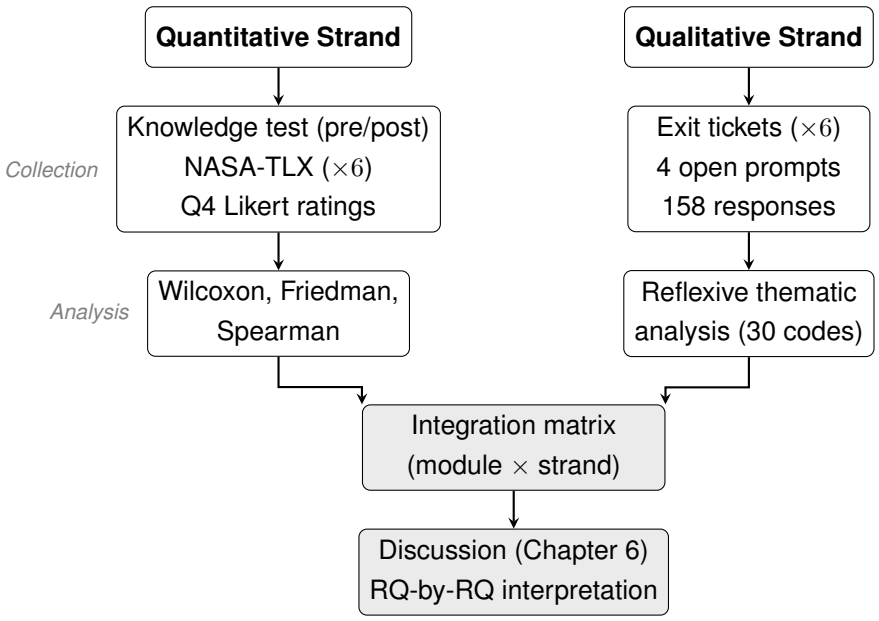


Figure 3.2.: Convergent parallel mixed-methods design after Creswell et al. (2017). Quantitative and qualitative strands ran concurrently during the workshop, were analyzed independently, and merged via an integration matrix in Section 5.7.

The quantitative pipeline finished within a week of the workshop. Qualitative coding took longer: 158 responses were tagged across a 30-code codebook during a separate analysis pass that ran through late February. Because the two timelines did not overlap, each analysis proceeded on its own terms. An integration matrix then mapped qualitative themes onto the per-module quantitative findings (Section 5.7), and the discussion chapter pulls from both strands for each research question. Late integration was not a methodological preference so much as a scheduling reality, but it had the side benefit of keeping the qualitative codes free from anchoring to the numbers.

The design carries one hard boundary on the workshop day itself: no delayed post-test and no retention check were administered during the session. Follow-up interviews had been scheduled for two to three weeks after the workshop to capture retention and transfer evidence that same-day instruments cannot provide. Due to insufficient participant response, those interviews were not conducted; claims about long-term learning therefore rest on the workshop-day evidence only, and this boundary is taken up in Section 6.7.

3.3. Curriculum Development Process

The curriculum grew out of a question that predated the thesis: could a learner who knows basic Python move from pixel-level array manipulation to sampling a denoising diffusion model, and could that progression hold together as a single course? Fifteen modules were planned to answer it. Development ran from October 2025 through early February 2026, with

the researcher as sole author, and produced a browser-based HTML site created with the Sphinx code documentation system, deployed via GitHub Pages.[1]

The fifteen modules span four tiers, summarized in Table 3.2. By the workshop date, completion levels ranged from production-ready exercises in several foundational modules to outright placeholders in the integration tier. Section 3.3.3 explains which modules were selected for testing and why.

Table 3.2.: Fifteen-module curriculum organized by tier.

Tier	Modules	Content Areas
Foundational	0–3	Definitions, Pixel Fundamentals, Geometry, Transformations
Intermediate	4–7	Fractals, Simulation, Noise, Classical ML
Advanced	8–9, 12	Animation, Neural Networks, Generative AI
Integration	10–11, 13–14	TouchDesigner, Interactive Systems, AI Integration

3.3.1. Pedagogical Framework Selection

Not all modules teach the same type of content. A color-mixing exercise and a derivation of the diffusion forward process need different instructional shapes, as Sweller et al. (1998) discussed in terms of element interactivity. Three frameworks were defined to handle that range, and the choice of which framework to apply to a given module followed from the content's theory-to-practice ratio.

Hands-On Discovery (Hands-On Discovery (HOD)) targeted topics where the learning objective was a visible procedure: creating an RGB gradient, evolving a cellular automaton, drawing a recursive fractal. The template opened with a Quick Start (run a working script, see the output), moved to two or three Core Concepts, and closed with a three-tier Guided Practice sequence. That sequence asked learners first to execute an existing script, then to modify its parameters, then to rewrite a key section from scratch, mirroring the gradual release of responsibility model (Pearson et al. 1983). Reading sat at roughly 25% of module time. The remaining 75% was spent in the editor.

Conceptual Deep-Dive (CDD) (Conceptual Deep-Dive) handled the opposite case: convolution kernels, perceptron weight updates, the noise schedule behind a diffusion model. Here, running code before understanding the mathematics would have produced confusion rather than insight. Conceptual Deep-Dive (CDD) modules shared the same exercise sequence as Hands-On Discovery (HOD) (Execute, Modify, Create) but front-loaded more Core Concept sections (three to four per module rather than two to three), each substantially more theory-dense, interleaving mathematical notation with worked examples before the learner wrote any code. Notation was introduced one symbol at a time, and each concept section included an implementation step so that theory never ran far ahead of practice. The reading-to-coding balance sat closer to 50/50.

1. Curriculum hosted at https://burakkagann.github.io/Pixels2GenAI/. Source repository at https://github.com/burakkagann/Pixels2GenAI.

The contrast between the two frameworks was deliberate. Hands-On Discovery (HOD) minimized extraneous load by letting visual feedback carry the explanation; Conceptual Deep-Dive (CDD) accepted a higher intrinsic load ceiling because certain topics cannot be simplified below a mathematical threshold without losing accuracy (Sweller et al. 1998). A third framework, Project-Based Synthesis (Project-Based Synthesis (PBS)), was planned for the capstone module but did not reach a testable state by the workshop date. Its absence from the empirical evaluation is noted in Section 3.1. Table 3.3 contrasts the three frameworks.

Table 3.3.: Comparison of the three pedagogical frameworks.

Feature	**HOD**	**CDD**	**PBS**
Content type	Visible procedures (gradients, automata, fractals)	Mathematical / theoretical (kernels, perceptrons, diffusion)	Capstone synthesis
Structure	Quick Start → Core Concepts (2–3) → Execute / Modify / Create	Framing statement → Core Concepts (3–4, theory-dense) → Execute / Modify / Create	(Not developed)
Reading : Coding	∼25 : 75	∼50 : 50	—
Load strategy	Minimize extraneous load via visual feedback	Accept higher intrinsic load ceiling	—
Workshop modules	M1, M2, M4	M3, M5, M6	None tested

3.3.2. Technology Stack and Delivery

Every exercise ran on Python 3.11. Keeping the workshop stack short was intentional; each additional package is a potential installation failure, and such failure generates frustration that has nothing to do with the learning content.

Workshop dependencies (M1–M6):

- NumPy and Pillow — array and image operations
- SciPy — kernel convolutions
- Matplotlib — plotting
- PyTorch + Hugging Face Diffusers — diffusion model training and sampling (M6 only)

Broader curriculum (not tested in workshop):

- OpenCV — computer vision tasks
- scikit-learn — classical machine learning
- Pygame — animation
- Noise-generation libraries — procedural content
- librosa — audio processing

These broader packages appear in the project's requirements file but were not installed on workshop machines. The curriculum site itself was built as an HTML site using the

Sphinx code documentation system and hosted as static HTML, so participants read browser-rendered documentation while coding in a local Visual Studio Code environment.

3.3.3. Module Selection for Workshop Testing

Six modules were selected to form a representative vertical slice through the fifteen-module curriculum, enabling participants to traverse the full progression from introductory array operations to generative model sampling within a single workshop day. Three used Hands-On Discovery (HOD) (M1: RGB Basics, M2: Cellular Automata, M4: Fractal Square) and three used Conceptual Deep-Dive (CDD) (M3: Convolution, M5: Perceptron, M6: DDPM Basics), producing an even framework split suited to direct comparison. In the full curriculum's numbering, the six spanned Module 1 (pixel fundamentals) through Module 12 (generative AI), covering the range from color gradients to diffusion sampling. The vertical-slice design required skipping five intermediate modules (Modules 0 and 5 through 8) between the fractal recursion of M4 (drawn from Module 4) and the perceptron training of M5 (drawn from Module 9). That gap was the price of reaching the advanced tier within a single day; its consequences for perceived coherence are discussed in Section 3.4.1. Chapter C documents the full module inventory, including completion status at the time of the workshop. Section 3.4.1 describes the sequencing and its consequences for participant experience.

3.4. Workshop Design

The workshop ran on February 12, 2026, as a single session lasting roughly seven hours, including a lunch break and short breaks taken after each module completion, a format comparable in structure to the two-day Carpentries workshops evaluated by Jordan et al. (2018), though compressed here into a single day. Nine participants worked through six curriculum modules in a fixed sequence, completing two data collection instruments after each one. The researcher's thesis supervisor was present throughout as co-facilitator, assisting with organizational logistics, troubleshooting technical errors, and recording facilitator observation notes.

3.4.1. Module Selection and Sequencing

Six modules were drawn from the fifteen-module curriculum to cover its full difficulty range within the time budget of a single day. The selection balanced framework coverage and content breadth against implementation readiness. Three modules used the Hands-On Discovery (HOD) framework (M1: RGB Basics, M2: Cellular Automata, M4: Fractal Square) and three used Conceptual Deep-Dive (CDD) (M3: Convolution, M5: Perceptron, M6: DDPM Basics). That gives an even split for framework comparison. In terms of the full curriculum's numbering, the six spanned Module 1 (pixel fundamentals) through Module 12 (generative

AI), covering introductory through advanced content. In practice, a participant who completed all six moved from creating a colored gradient (M1) to sampling a trained diffusion model (M6) over the course of roughly seven hours, including breaks. M1 was the natural entry point: it requires no programming prerequisites beyond running a script, and its visual output provides unambiguous feedback. M6 anchored the far end because it reaches the curriculum's final destination: training and sampling from a generative model. The four modules between them filled the progression; M2 adds rule-based generation, M3 applies kernel computation to images, M4 introduces recursion, and M5 bridges to neural network architectures. The Project-Based Synthesis (PBS) framework was excluded for the reasons discussed in Section 3.1. Section 4.6 provides full descriptions of each module.

The alternating pattern of Hands-On Discovery (HOD) and Conceptual Deep-Dive (CDD) modules was partly deliberate and partly inherited from the curriculum structure. Convolution mathematics requires conceptual setup before practice makes sense, and the same holds for perceptron training and diffusion processes; all three topics belong naturally to Conceptual Deep-Dive (CDD). The resulting sequence, Hands-On Discovery (HOD)-Hands-On Discovery (HOD)-Conceptual Deep-Dive (CDD)-Hands-On Discovery (HOD)-Conceptual Deep-Dive (CDD)-Conceptual Deep-Dive (CDD), created a rough alternation of lower-demand and higher-demand modules across the session.

The ordering followed the curriculum's built-in progression with one deliberate compression. M1 through M4 proceeded in curricular order: color basics, cellular automata, convolution, fractal recursion. Each built on array manipulation skills from the preceding module. Between M4 (Fractal Square, drawn from Module 4) and M5 (Perceptron, drawn from Module 9), four intermediate modules were skipped. That jump was the price of reaching the advanced tier within a single day. Its consequences for perceived coherence surface in the results: participants reported weaker connections to earlier content starting at M5 (Section 5.6), and the Generative AI section of the knowledge test produced the weakest gains (Section 5.2).

Despite the gap, the six modules were not independent lessons. Each reused and extended specific concepts introduced earlier, forming a scaffolding chain that carried participants from basic array indexing to generative model sampling. Figure 3.3 traces the key concept dependencies across the workshop sequence. M1 (RGB Basics) established arrays as images, together with indexing, slicing, and element-wise operations. M2 (Cellular Automata) reused indexing to implement neighborhood lookups and added iteration and emergence. M3 (Convolution) generalized neighborhoods into weighted kernels and introduced feature detection. From M3 the chain branched: M4 (Fractal Square) drew on recursive array construction and self-similarity, while M5 (Perceptron) extended kernels into learnable weights and added a training loop with an activation function. M6 (DDPM Basics) sat at the convergence point, relying on the array-as-image representation from M1, the iterative update logic from M2, the convolutional structure from M3, and the optimization loop from M5. These dependencies meant that the conceptual distance between adjacent workshop modules grew with each step, a pattern reflected in the cognitive load data reported in Section 5.3.

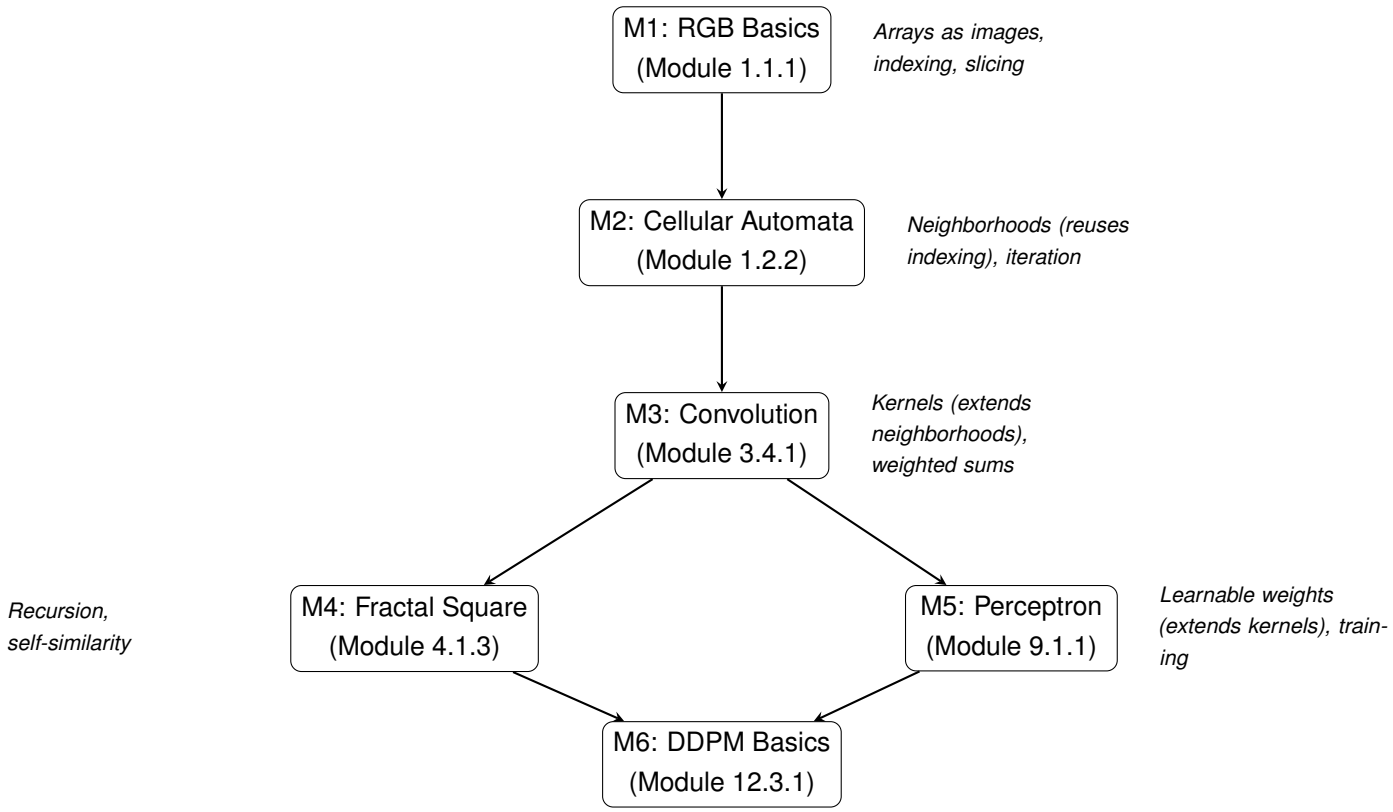


Figure 3.3.: Conceptual scaffolding chain across the six workshop modules. Arrows indicate that the target module reuses or extends a concept introduced in the source module. M6 (DDPM Basics) sits at the convergence point, drawing on array representations (M1), iterative processes (M2), convolutional structures (M3), and optimization loops (M5).

3.4.2. Session Protocol

Table 3.4 outlines the session timeline. Participants brought their own laptops. Prior to the workshop, the researcher shared setup instructions covering Python 3.11 installation, Visual Studio Code configuration, and the required dependencies (NumPy, Pillow, SciPy, Matplotlib, and, for M6, PyTorch plus the Diffusers library). On the day itself, each participant also received a USB flash drive containing a dedicated HTML site (built with Sphinx) prepared specifically for the six workshop modules; the offline build ensured that all documentation and exercise files were accessible without internet connectivity. The session opened with a 24-item multiple-choice pre-test (Section 3.6), which took approximately fifteen minutes.

Participants then worked through the six modules in order. After completing each module, they filled out a raw NASA-TLX survey (six subscales rated on a 21-point scale) followed by a five-item exit ticket containing four open-ended prompts and one Likert-scale self-assessment of understanding. A short break followed each module completion. A lunch break of approximately 45 minutes separated M3 from M4. That cycle of module, instruments, and break repeated six times. A parallel 24-item post-test closed the session.

Table 3.4.: Workshop session protocol, February 12, 2026.

Stage	Activity	Framework	Est. Duration
0	Pre-test (24 MC items)	—	15 min
1	M1: RGB Basics + TLX + Exit Ticket + Break	Hands-On Discovery (HOD)	30 + 5 min
2	M2: Cellular Automata + TLX + Exit Ticket + Break	Hands-On Discovery (HOD)	30 + 5 min
3	M3: Convolution + TLX + Exit Ticket + Break	Conceptual Deep-Dive (CDD)	40 + 5 min
	Lunch break		$\sim$45 min
4	M4: Fractal Square + TLX + Exit Ticket + Break	Hands-On Discovery (HOD)	30 + 5 min
5	M5: Perceptron + TLX + Exit Ticket + Break	Conceptual Deep-Dive (CDD)	50 + 5 min
6	M6: DDPM Basics + TLX + Exit Ticket	Conceptual Deep-Dive (CDD)	40 + 5 min
7	Post-test (24 MC items)	—	15 min

Note. A short break followed each module completion. Durations are targets; participants advanced at their own pace. Individual completion times were not recorded.

Each module had a target duration ranging from 30 minutes (M1, M2, M4) to 50 minutes (M5), but participants set their own pace. The absence of individual completion timestamps is acknowledged as a limitation in Section 6.7.

3.4.3. Facilitation and Technical Setup

The researcher and the thesis supervisor circulated throughout the session, answering technical questions and troubleshooting when software failed to run. No lectures or live demonstrations were given. Participants read instructions from the curriculum's browser-based HTML documentation and ran Python scripts in their local VS Code environment. The supervisor assisted with organizational logistics, helped resolve dependency errors on individual machines, and recorded structured facilitator observation notes. One purpose of the hands-off instructional approach was to test whether the modules could stand alone without sustained instructor support (cf. Kirschner et al. (2006), who argue that minimal guidance fails learners with low prior knowledge), a question relevant to the curriculum's use beyond a single classroom.

Setup difficulties appeared from the first module onward. Participants with no prior Python experience struggled with IDE navigation, dependency installation, and running scripts in M1; the flash drive's pre-built environment resolved most issues, but the initial learning curve was steeper than anticipated for complete beginners. Problems were most concentrated in M6 (DDPM Basics), which required PyTorch; four of seven exit-ticket challenge responses for that module cited installation problems or hardware limitations as their primary obstacle.

One participant (P04) completed all six modules and all six TLX surveys but did not submit exit tickets from M2 onward and did not complete the post-test, reducing paired knowledge analyses to $n = 8$. P06 did not complete exit tickets for M5 and M6. NASA-TLX data were 100% complete across all nine participants and all six modules. Section 5.1 reports the full data completeness picture.

3.5. Participants

Nine participants, coded P01 through P09, took part in the workshop. Recruitment ran through personal and university networks during January 2026: the researcher circulated an open call targeting adults with at least some exposure to Python. Everyone who responded was accepted. Nine signed up. The invitation listed basic Python familiarity as a prerequisite, but two participants arrived with no prior Python experience, and neither was turned away. The call reached a narrow pool of personal contacts rather than a broad public audience, making this a convenience sample (Creswell et al. 2017). Volunteers who signed up were likely more motivated than a representative population of learners; that self-selection caveat applies to every finding reported here.

The sample profile was as follows:

- **Age:** 18–44
- **Gender:** 5 male, 4 female
- **Education:** 4 master's degrees, 2 bachelor's, 3 high school
- **Programming experience:** 4 ($<$1 yr), 2 (1–2 yr), 2 (3–5 yr), 1 (10+ yr)
- **Python proficiency:** 2 never used, 4 beginner, 2 intermediate, 1 advanced
- **Composite experience score:** $M = 0.209$, $SD = 0.162$ (0–1 scale; squarely beginner territory; only one participant scored above the midpoint)

Table 5.1 in the Results chapter provides the full breakdown.

Before the pre-test, each participant reviewed a one-page consent document (Section A.2) that named the six instruments, estimated a seven-hour session, and stated that departure at any point carried no penalty, following the informed consent guidelines in Creswell et al. (2017). P04 exercised that option in part, completing all modules but declining exit tickets from M2 onward and the post-test. All responses were recorded under the P01 through P09 codes, with a separate file linking codes to names stored on an encrypted drive. Only the researcher had access. That link file will be destroyed once the thesis is accepted. No names appear in this document.

P06 completed the full session but did not submit exit tickets for M5 or M6. Combined with P04's missing post-test and exit tickets, these gaps drop paired knowledge analyses to $n = 8$ and leave qualitative response counts varying by module. TLX data had no missing values. Across all instruments, 127 of 135 cells were filled. Section 3.7 addresses how the remaining gaps were handled.

3.6. Data Collection Instruments

All six instruments from the chapter opening are reproduced in Chapter A.

3.6.1. Knowledge Assessment Test

The knowledge test comprised 24 multiple-choice items split across four content sections, each mapped to a pair of workshop modules (Table 3.5). Pre-test and post-test used parallel forms. Both covered the same conceptual ground, but the questions were worded differently to limit memorization between sittings. Every item had four answer choices plus an "I Don't Know" option; selecting "I Don't Know" scored the same as a wrong answer. Each item was scored 1 for correct, 0 for anything else.

Table 3.5.: Knowledge test content sections and workshop module mapping.

Section	Content Area	Items	Mapped Modules
A	Arrays & Images	6	M1 (RGB Basics), M2 (Cellular Automata)
B	Computational Concepts	6	M3 (Convolution), M4 (Fractal Square)
C	Neural Networks	6	M5 (Perceptron)
D	Generative AI	6	M6 (DDPM Basics)

That "I Don't Know" option served two purposes. For participants, it took away the incentive to guess on unfamiliar material. For the analysis, it produced a trackable marker of knowledge boundaries: the drop in "I Don't Know" selections from pre-test to post-test became a finding in its own right. Section D (Generative AI) had the highest "I Don't Know" rate at both time points.

Each content section mapped onto a pair of workshop modules, with items written to track the specific learning objectives of those modules. No outside expert reviewed the test. The researcher wrote and checked every item against the relevant module content alone. Reliability data show up in Section 5.2: the post-test reached KR-20 $= 0.913$, while the pre-test managed only 0.742, weighed down by floor effects when most participants chose "I Don't Know."

3.6.2. NASA Task Load Index

Cognitive load was tracked through the Raw NASA Task Load Index (Hart et al. 1988), administered without the weighting step from the original protocol (Hart 2006). In the weighted version, participants rank subscale importance after every administration; running that card-sorting procedure six times would have eaten into the session. The raw format skips it. Participants rated six subscales directly: Mental Demand, Physical Demand, Temporal Demand, Performance, Effort, and Frustration. The rating scale ran from 0 to 100 in increments of five (21 gradations), converted to 1 through 20 for analysis. Performance was the only subscale where lower raw scores mean better outcomes, so it was flipped: after inversion, higher values on every subscale point toward greater load.

Each participant filled out a TLX form after every module: six per person, 54 total. The repeated-measures structure makes the Friedman test the right omnibus comparison for workload variation across the sequence (Section 3.7).

3.6.3. Exit Tickets

Nine participants, four open prompts, six modules: the theoretical ceiling was 216 text entries. In practice, attrition and blanks brought the usable count to 158. Each exit ticket went out alongside the TLX form for that module and contained five items (Section A.6 reproduces the full instrument):

1. **Q1:** What did you learn? (open-ended)
2. **Q2:** How does this connect to earlier modules? (open-ended)
3. **Q3:** What was most confusing? (open-ended)
4. **Q4:** Self-rated understanding (Likert scale, 1–5)
5. **Q5:** What would you like to explore next? (open-ended)

Some entries were one word. Others ran to three or four sentences. That corpus supported the thematic analysis in Section 3.7 but was too thin for any frequency-based quantitative treatment of the codes. Q4, being numerical, was analyzed alongside the TLX data.

3.6.4. Additional Instruments

Two instruments sat outside the per-module cycle. A demographics questionnaire (Section A.3), filled out before the session, covered age, gender, education, programming history, and self-rated proficiency across Python, NumPy, and six domain areas. Eight of those responses were normalized into the 0–1 composite score reported in Section 3.5. During the session itself, the researcher kept structured notes via a facilitator observation protocol (Section A.7), logging engagement levels, help requests, and technical failures; those notes were not coded systematically before the submission deadline. The follow-up interview, scheduled for two to three weeks post-workshop, drew no sufficient responses and was not conducted. A brief post-workshop evaluation form went out at session close; seven of the nine participants returned it, and their responses appear in Section 5.7 as supplementary descriptive data.

3.7. Data Analysis

With only nine participants ($n = 8$ for paired comparisons), parametric tests were off the table. TLX subscale ratings are ordinal, not interval, and no distribution built from nine data points can be trusted to meet normality assumptions. Every statistical test reported in Chapter 5 is non-parametric (Siegel 1956).

Three test families covered the quantitative work. Pre-post knowledge gains were tested with Wilcoxon signed-rank comparisons, both for the 24-item total and for each of the four content sections separately. Whether TLX ratings varied across the six modules was a

Friedman-test question, run once per subscale. Relationships between continuous variables (for instance, composite experience versus average workload) were checked with Spearman correlations. Each test was paired with an effect size measure: Cohen's d for the paired comparisons (Cohen 1988), Kendall's W for concordance, and Hake's normalized gain (g) for knowledge outcomes (Hake 1998). Where a Friedman test came back significant, pairwise Wilcoxon comparisons followed across all 15 module pairs, with the significance threshold Bonferroni-corrected to $\alpha_{\text{adj}} = .0033$ (Dunn 1961). Exact p-values were used throughout; asymptotic approximations are unreliable at $n = 9$. The computational work ran in Python 3.11 with SciPy and pandas. Table 3.6 summarizes the mapping from analysis goals to statistical procedures.

Table 3.6.: Summary of analysis methods.

Analysis Goal	**Test**	**Effect Size**	**Correction**	**RQ(s)**
Pre-post knowledge gains	Wilcoxon signed-rank	Cohen's d, Hake g	—	1, 4, 5
Workload across modules	Friedman	Kendall's W	Bonferroni	2
Variable relationships	Spearman correlation	r_s	—	2
Qualitative themes	Reflexive thematic analysis	—	—	All

Open-ended exit ticket responses were coded following the thematic analysis framework of Braun et al. (2006), later termed *reflexive* thematic analysis (Braun et al. 2019). The researcher read all 158 responses twice before assigning any codes, then built codes inductively across two full passes. No categories existed beforehand. The two passes produced a 30-code codebook organized under five headings: Learning (5 codes), Connections (6 codes), Challenges (10 codes), Engagement (6 codes), and Experience (3 codes). A single response could receive more than one code; the final count was 204 code applications across all responses.

All coding was done by one person: the researcher. No second analyst checked the code assignments, and no agreement statistic (such as Cohen's κ) was computed. Under the reflexive thematic analysis framework (Braun et al. 2019), coding is an act of interpretation rather than a standardized measurement, which means inter-rater agreement is not a formal requirement (Braun et al. 2006). That rationale does not remove the limitation. A single coder's blind spots go undetected. Coding wrapped up in late February 2026, about two weeks after the last quantitative result was finalized.

The numbers came first. Qualitative coding started only after all statistical results were locked. That sequencing was half scheduling, half methodological choice: it kept the codes free from anchoring to the numbers. An integration matrix brought both strands together afterward. Each module occupied a row; the columns paired TLX scores, knowledge gains, and Q4 understanding ratings with the dominant qualitative codes for that module. Where

both strands pointed the same way, the cell was marked as convergent; where they pulled apart, divergent. Chapter 6 works through each research question using both strands.

P08 raised a flag during data cleaning. Four of six TLX subscales sat at the scale floor across all six modules, suggesting a possible satisficing pattern. To check, every key analysis was re-run without P08. Nothing changed. The Friedman test for overall workload still held ($\chi^2(5) = 14.98$, $p = .010$), and the experience-load correlation tightened slightly ($r_s = -.893$, $p = .007$). Two validity checks rounded out the analysis. The knowledge test's internal consistency was assessed with KR-20 (Section 5.2). For convergent validity, Q4 self-ratings were correlated with TLX composites; pooled across all module-participant pairs, the Spearman coefficient came to $r_s = -.514$ ($p < .001$), supporting the reading that perceived understanding and perceived workload sit at opposite ends of one continuum. Across all instruments, 127 of 135 data cells were filled.

3.8. Use of Generative AI Tools

Two generative AI tools supported various stages of this research. Their use is disclosed here in accordance with the SRH guidance framework on generative AI in academic work (Version 1.0, July 2025). Neither tool produced content that was adopted without modification; every output was checked against primary sources or raw data, revised where necessary, and integrated only after the researcher had confirmed its accuracy.

Curriculum design. During the early design of the fifteen-module curriculum, Claude Opus 4.6 (Anthropic 2026), a large language model developed by Anthropic, was used to brainstorm candidate module structures, identify potentially relevant learning theories for each unit, and outline the skeleton chapter structure of the thesis. Each suggestion was weighed against the pedagogical literature before being kept or set aside.

Literature search. Elicit (Elicit 2026), an AI-powered research assistant, helped locate candidate papers aligned with the five research questions. Results were cross-checked in Google Scholar and Scopus, and only papers whose relevance and quality held up under direct reading entered the final reference list.

Instrument development. Claude Opus 4.6 assisted with brainstorming the structure of the participant feedback form and the facilitator observation form used during the workshop. Once the researcher had finalised the content and layout of each instrument, the model generated printable PDF versions. The pre- and post-test knowledge items were also validated with the tool to confirm that every question tested only material explicitly presented in the curriculum modules.

Technical development. Claude Opus 4.6 also assisted with debugging the Python scripts that processed workshop data, troubleshooting exercise code in the curriculum modules, formatting the deployed Sphinx documentation site, and generating the QR code that links to the curriculum platform. The LaTeX source for the thesis figures and tables, including the data-visualisation charts in the results chapter (trajectory plots, heatmaps, correlation matrices,

and box plots), was produced with the tool after the researcher supplied the underlying data and the desired visual structure; each output was inspected and revised before inclusion.

Data verification and statistical reporting. Once every statistical test had been computed manually in Python, the same model was used to cross-check reported numbers, flag potential inconsistencies between the results and discussion chapters, and probe whether the interpretations attached to each test outcome held up under scrutiny. Formatting the statistical results in APA-compliant LaTeX syntax (e.g. Wilcoxon, Friedman, and Spearman tests) and drafting the accompanying interpretive sentences were likewise carried out with AI assistance; the researcher verified every value against the raw output before accepting the final wording. The abbreviation list was compiled with similar assistance. All of these checks were confirmatory; no analytical decision originated from the tool.

The design of the study, the analysis of the data, and every argument advanced in this thesis remain the researcher's own work. A complete list of AI tool usages appears in the addendum at the end of this document.

4. Design and Implementation

At one end of the curriculum, a script colors individual pixels in a NumPy array. At the other end, a diffusion model generates images from noise. Fifteen modules bridge those two points, progressing through noise generation, simulation and emergent behavior, classical image processing, animation, and neural network training, among other topics. The curriculum sits at the center of the Design-Based Research cycle outlined in Section 3.1 and is publicly hosted at https://burakkagann.github.io/Pixels2GenAI/. Six of those fifteen reached workshop-ready status and were tested on February 12, 2026 (Chapter 3). The other nine were not tested. At the time of the workshop, the tested modules had working code with complete documentation and scaffolded exercises; Modules 10, 11, and 13, which span TouchDesigner fundamentals, interactive systems, and AI–TouchDesigner integration, existed only as structural outlines, as did Modules 5 through 8 and 14.

Two commitments shaped every module. The first is visual output. Each exercise produces a visual result that makes the underlying computation tangible. When a participant writes a 3×3 convolution kernel, the blurred photograph appears on screen in seconds; adjusting a perceptron's learning rate redraws the decision boundary across training epochs. Immediate feedback is the point. Dual coding theory (Paivio 1986) supplies the rationale: when code and its visual output appear together, learners encode the concept through both verbal and imagistic systems. The broader multimedia research supporting this choice is reviewed in Section 2.2. The second commitment is progressive complexity. Module 1 asks only for array indexing. Module 12 assumes familiarity with probability distributions, gradient computation, and noise scheduling. The curriculum bridges that gap by chaining prerequisites: each module adds one or two new ideas while everything else stays familiar. Arrays lead to kernels; kernels lead to convolution; convolution leads to feature maps; feature maps lead to neural network layers. Sweller et al. (1998) would describe this as keeping element interactivity low; when most task components are already rehearsed, working memory has room for the few that are novel. Whether that logic holds under actual workshop conditions is tested in Chapter 5; the present chapter lays out the design rationale.

Content was packaged into exercises through one of three pedagogical frameworks. Hands-On Discovery (HOD) modules front-load a working code example, let participants run and modify it, and only formalize the underlying theory afterward. The theory-to-practice ratio sits at roughly 25:75. Conceptual Deep-Dive (CDD) modules share the same structural skeleton but pack in more numerous and theory-dense Core Concept sections, interleaving mathematical notation with worked examples before each coding step. Their theory-to-practice ratio is closer to 50:50. A third framework, Project-Based Synthesis (PBS), targets

capstone-style integration tasks but was not ready for testing by February 2026 (Section 3.1 notes this gap). Only Hands-On Discovery (HOD) and Conceptual Deep-Dive (CDD) went into the workshop, and their contrasting cognitive load profiles turned out to be one of the study's clearest empirical findings.

What follows describes the full fifteen-module structure (Section 4.1), the two frameworks tested in the workshop (Sections 4.2 and 4.3), and the assessment instruments (Section 4.4). The technology stack (Section 4.5) and the six individual workshop modules (Section 4.6) close the chapter.

4.1. Curriculum Architecture

Table 4.1 lists all fifteen modules in their intended teaching order together with the sub-modules that comprise each one. Numbering starts at zero because Module 0 covers prerequisite vocabulary and tool installation rather than visual programming as such; from Module 1 onward, every exercise produces at least one visual artifact that the learner can inspect, modify, and rebuild.

Table 4.1.: Fifteen-module curriculum sequence with framework assignments and sub-module breakdown.

#	Module Title	Fw.	Sub-modules
0	Foundations & Definitions	CDD	What Is Generative Art; Defining AI/ML; Images As Data; Setup
1*	Pixel Fundamentals	HOD	Color Basics; Pixel Patterns; Structured Compositions
2*	Geometry & Mathematics	HOD	Basic Shapes; Coordinate Systems; Mathematical Art
3*	Transformations & Effects	HOD	Geometric Transforms; Masking & Compositing; Artistic Filters; Signal Processing
4*	Fractals & Recursion	HOD	Classical Fractals; Natural Fractals; L-Systems
5	Simulation & Emergence	HOD	Particle Systems; Flocking & Swarms; Physics Simulations; Growth
6	Noise & Procedural Generation	HOD	Noise Functions; Terrain Generation; Texture Synthesis; Wave Patterns
7	Classical Machine Learning	CDD	Clustering; Classification; Dimensionality Reduction; Statistical Methods
8	Animation & Time	HOD	Animation Fundamentals; Organic Motion; Cinematic Effects; Generative Animation
9*	Neural Networks	CDD	NN Fundamentals; Architectures; Training Dynamics; Feature Visualization
10	TouchDesigner Fundamentals	CDD	TD Environment; Recreating Exercises; NumPy-to-TD Pipeline; Interactive Controls
11	Interactive Systems	HOD	Input Devices; Computer Vision in TD; Physical Computing; Network Communication
12*	Generative AI Models	CDD	GANs; VAEs; Diffusion Models; Language Models; Personalization; Transformers; Frontiers
13	AI + TD Integration	CDD	ML Models in TD; Real-time AI Effects; Generative Models Live; Hybrid Pipelines
14	Data as Material	HOD	Data Sources; Visualization; Sonification; Physical Data Sculptures

Note. HOD = Hands-On Discovery; CDD = Conceptual Deep-Dive. * = contributed one exercise to the February 2026 workshop (Section 4.6). The **Fw.** column shows the default framework assigned at the *module* level; individual exercises within a module may carry a different assignment. Module 3 (default: HOD) contributed a CDD exercise (Convolution), and Module 4 (default: HOD/CDD hybrid) contributed a HOD exercise (Fractal Square). These exercise-level assignments are documented in Section 4.6.

The sequence splits into three broad bands. Modules 0 through 4 deal with image representation and spatial operations. A learner entering Module 1 assigns RGB triplets to individual array coordinates, coloring pixels one at a time. Later sections of the same module introduce neighbor-counting rules: each cell updates according to how many of its eight neighbors are active, and spatial patterns emerge from those purely local decisions. Geometry takes over in Module 2. The learner draws lines, circles, and gradient fields by evaluating coordinate expressions across the pixel canvas, then constructs mathematical curves such as spirals

and rose curves by sweeping parametric equations through the grid. Module 3 formalizes spatial filtering through convolution: a small weight matrix slides across the image, producing blurred, sharpened, or edge-detected output depending on the kernel values. Module 4 closes the band with fractal recursion, a function that calls itself to stamp progressively smaller squares into a canvas. By this point the learner has rehearsed array indexing, neighbor logic, coordinate arithmetic, element-wise computation, and recursive function design, all through tasks whose visual results appeared within seconds of pressing *Run*.

Modules 5 through 8 shift from static pictures to dynamic processes. Simulation exercises introduce particle systems, flocking algorithms, and physics-based growth models whose large-scale structure emerges from purely local update rules applied over hundreds of iterations. Procedural noise functions replace uniform randomness with spatially coherent textures, a stepping stone to the stochastic schedules that diffusion models use later in the curriculum. Classical machine learning enters at Module 7 with clustering algorithms that partition pixels by color similarity, and Module 8 adds a time axis by stringing single frames into animations. The decisive change in this band is that outputs now evolve across iterations or time steps, which forces the learner to track state inside loop structures. Training loops in the neural network modules rely on exactly that skill.

Modules 9 through 14 target neural networks and generative models. Module 9 builds a single-layer perceptron on a synthetic two-class dataset, plotting the decision boundary as it moves across training epochs. Later exercises in the same module scale to multi-layer architectures and convolutional feature extractors. Modules 10 and 11 were planned to bridge Python-trained models into TouchDesigner for real-time visual output; both existed only as structural outlines at the time of the workshop. Module 12 represents the most advanced content developed to date, surveying generative adversarial networks, variational autoencoders, diffusion models, personalization techniques, and emerging architectures such as flow matching. By this point the pixel arrays first created in Module 1 serve as training data for models that learn to synthesize new images from noise. Modules 13 (AI-TouchDesigner integration) and 14 (data-driven art) had not moved past placeholder stage by February 2026.

Prerequisite chaining ties the three bands into one continuous path. No module asks the learner to absorb more than two genuinely new concepts; everything else in the task draws on ideas already practiced. Module 3 makes the pattern concrete: it takes array indexing and shape drawing from Modules 1–2 as given, then adds only the kernel abstraction and the sliding-window mechanism. Module 9 reuses weighted sums (first met through convolution kernels), iterative updates (from simulation), and array broadcasting, then layers gradient descent and activation functions on top. The logic maps directly to element interactivity in Sweller et al. (1998): when most components of a problem have been automated through practice, the few remaining novel components sit inside working-memory limits. Packing three or four unfamiliar ideas into a single module would force the learner to juggle multiple unintegrated elements at the same time, precisely the overload condition that Sweller (1988) ties to excessive cognitive load.

Six exercises, pulled from six of these fifteen modules, formed the workshop evaluated in this thesis. The selection aimed for a vertical slice through the full sequence: Module 1 sits at the foundation level, Module 12 at the advanced endpoint, and Modules 2, 3, 4, and 9 fill the space between them. Three exercises used Hands-On Discovery (HOD) and three used Conceptual Deep-Dive (CDD), creating a balanced framework comparison. Because the six topics span pixel manipulation, cellular automata, convolution, fractal geometry, neural network training, and diffusion, performance gains measured across the set cannot be pinned to a single narrow skill domain. Section 4.6 describes each workshop exercise individually.

Framework assignment was driven by the conceptual density of each topic. Modules where a single running code example can carry the core mechanism were built on the Hands-On Discovery (HOD) template: execute the example, watch the output, adjust parameters, build something original. RGB color mixing, cellular automata, and fractal recursion all fit that mold. One line changed, one visible consequence. No extended theoretical preparation is needed before the feedback loop begins producing insight. Modules whose central ideas resist a single-example treatment received Conceptual Deep-Dive (CDD) instead. Convolution requires explaining kernels, the sliding-window operation, and why different weight matrices create different visual effects before any individual code block can carry meaning on its own. Neural network training calls for at minimum a cost function, a gradient rule, and a parameter update step. Diffusion models stack a forward noise schedule onto a learned reverse denoiser. For those topics, Conceptual Deep-Dive (CDD) opens with a motivating question and distributes theory across several code-and-explanation cycles, so that no single moment demands too much from working memory at once.

The assignment was not purely about difficulty, though. Module 5 (Simulation) covers particle systems, flocking algorithms, physics-based models, and growth patterns. The mathematics behind those topics runs deep, but the core insight is simple: local update rules produce large-scale structure. A swarm forming on screen makes that point faster than a page of coupled equations would. Module 5 therefore received Hands-On Discovery (HOD). Module 0 (Foundations), by contrast, covers relatively simple material (definitions, installations) but opens with “What is generative art?”, a question that calls for structured discussion before any code runs. It received Conceptual Deep-Dive (CDD) for that reason.

The final count is nine Hands-On Discovery (HOD) modules against six Conceptual Deep-Dive (CDD). That ratio tilts the curriculum toward hands-on practice, and the tilt is deliberate: where visual feedback is immediate and unambiguous, scaffolded exploration with embedded guidance can produce effective learning (Hmelo-Silver et al. 2007). The six Conceptual Deep-Dive (CDD) exceptions cluster where mathematics or algorithmic logic runs too deep for the output alone to teach the concept. A third template, Project-Based Synthesis (PBS), was designed for open-ended capstone projects pulling skills from several modules together, but no Project-Based Synthesis (PBS) content reached production quality before the February 2026 workshop. Sections 4.2 and 4.3 describe the internal scaffolding of each tested framework.

All content sits on an HTML site created with the Sphinx code documentation system, deployed as static HTML through GitHub Pages at https://burakkagann.github.io/Pixels2GenAI/. Sphinx was chosen because its reStructuredText markup keeps code blocks, mathematical notation (via MathJax), and embedded output images on a single page, binding instructional text to executable code without forcing the learner to switch contexts. The Py-Data theme supplies sidebar navigation, full-text search, and collapsible dropdown directives; exercise solutions stayed hidden until the learner chose to reveal them. Static HTML requires no server, so participants during the workshop loaded the site on personal laptops with nothing more than a browser. They copied code snippets into local Python sessions and alternated between two windows: the browser for instructions, the terminal for execution. That two-window arrangement kept every scaffolding element visible while the learner typed, a practical application of the split-attention reduction principle that Sweller et al. (1998) place at the center of extraneous load management.

4.2. Framework 1: Hands-On Discovery

The Hands-On Discovery (HOD) template puts visual output ahead of conceptual explanation. Every HOD exercise opens with a Quick Start segment: a short, complete code example that the learner runs within the first two to three minutes of the session. The RGB Basics exercise (M1 in the workshop) starts with eighteen lines of Python that create a 200×200 pixel canvas and split it into cyan and magenta halves. The learner sees the two-tone rectangle before reading a single line of theory about how NumPy arrays store images. That ordering is the template's defining feature.

After the Quick Start, a Core Concepts block covers between one and three new ideas, never more. In M1 the concepts are RGB channel structure and integer value ranges (0–255). In M2 (Cellular Automata) they are neighbor counting and rule-based state transitions. In M4 (Fractal Square) the concept is recursive subdivision. Keeping the count low is not accidental: every new concept competes for working-memory space alongside the syntax and the visual output the learner is still processing. Fewer novel elements mean less element interactivity, which Sweller et al. (1998) tie directly to reduced intrinsic load.

The third stage, Guided Practice, follows a four-step scaffolding sequence: Execute, Modify, Create, and an optional Make It Your Own challenge. Execute asks the learner to run a provided script and observe the output. Modify asks for targeted parameter changes: swap the kernel weights in a blur filter, for instance, then check whether the image gets sharper or softer. Create removes the supporting structure entirely; the learner writes a new function from scratch using the concepts just practiced. Make It Your Own calls for an original creative application, a generative artwork, an animation, or a data visualization designed without step-by-step guidance. The progression moves from full scaffolding to none, mirroring the worked-example fading sequence that Sweller et al. (1998) associate with effective instructional design. Each step is paired with a visible output, so the learner

can verify correctness by inspecting the image rather than parsing a stack trace or a numeric return value.

M4 (Fractal Square) applied the template to a more abstract concept: recursion. The Quick Start displayed a completed fractal, a square subdivided into progressively smaller squares in a self-similar pattern. The learner ran the code and saw the result before encountering the word “recursion.” Core Concepts then introduced recursive function calls using the fractal as the anchor example: the function draws a square, calls itself four times at reduced scale, and each call draws another square. The guided practice exercises asked the learner to run the base case, change the recursion depth, rewrite the function with a different subdivision rule, and finally design an original fractal. Recursion is typically taught through abstract examples like factorials and Fibonacci sequences, but here the visual output made the recursive structure tangible. Each call left a visible mark on the canvas, and increasing the depth added visible detail. The tight feedback loop that Hands-On Discovery (HOD) depends on held, even for a concept that usually lives in the abstract.

Two constraints kept every HOD exercise compact. First, the theory-to-practice ratio sat at roughly 25:75. Explanatory text was short and interleaved with code rather than front-loaded into a reading block. Second, every code segment produced an image. No exercise returned only a number or a print statement; output was always visual. That constraint put Paivio (1986) dual coding into practice: the code itself forms one memory trace (verbal channel), the on-screen image forms a second (imagistic channel), and the pairing strengthens retention of the underlying concept. Whether the lower theory load actually translated into lower cognitive demand during the workshop is tested in Chapter 5; the design intent was to keep the learner inside a feedback loop tight enough that mistakes became visible in seconds and corrections could be tested just as fast.

HOD exercises in the workshop ran between 25 and 35 minutes. M1 (RGB Basics) took approximately 30 minutes. M2 (Cellular Automata) finished in roughly the same window. M4 (Fractal Square) ran slightly longer because participants spent extra time experimenting with recursion depth, colour channel substitutions, and asymmetric compositions produced by selectively removing corner calls. Those compact time windows are a direct consequence of the 25:75 split: with less front-loaded reading, participants reached the coding exercises quickly and spent the bulk of each session writing and running code.

4.3. Framework 2: Conceptual Deep-Dive

The Conceptual Deep-Dive (CDD) template exists because certain topics resist the single-example approach that Hands-On Discovery (HOD) relies on. When a learner runs a diffusion model and watches an image emerge from noise, the output looks impressive. It is also opaque. The pixels say nothing about why noise was added in the first place, and the variance schedule that governed the corruption is invisible in the final image. Conceptual Deep-Dive

(CDD) addresses the gap by front-loading conceptual scaffolding before the learner writes any code.

Every CDD exercise opens with a framing statement that anchors the upcoming theory in a concrete task. M3 (Convolution) opened by telling the learner that a small grid of numbers, slid across every pixel, is all it takes to blur, sharpen, or outline a photograph. The M5 (Perceptron) hook took a different angle, starting from Rosenblatt's 1958 architecture and promising a working binary classifier in about 30 lines of NumPy. M6 (DDPM Basics) opened with the claim that diffusion models displaced GANs as the dominant generative architecture within two years of their introduction. None of these hooks posed a question. They told the learner what to expect and why the theory ahead of them was worth the reading time.

After the opening hook, three or four Core Concept sections lay out the theoretical material the exercises will rely on. Each section pairs an explanation with a worked code example that the learner reads but does not yet modify. M5 illustrates the pattern. The first of M5's three Core Concept sections lays out the perceptron architecture, from weighted inputs through a bias term to a step-function threshold. A second section supplies the forward-pass code, built around `np.dot(weights, x) + bias`, and a third introduces the learning rule with a code block that updates weights across several training epochs. The learner therefore has all three building blocks on screen, as worked reference code, before writing anything original. That sequencing is the structural difference from Hands-On Discovery (HOD): the learner works through more conceptual ground before touching a single `TODO` block (Sweller et al. 1998).

The exercises themselves follow the same Execute, Modify, Create sequence used in Hands-On Discovery (HOD). In M5, Exercise 1 ran a pre-built perceptron and posed reflection questions about convergence speed and boundary geometry. Exercise 2 pushed parameters past safe limits: slowing the learning rate to 0.01 made convergence visibly sluggish, and widening cluster spread to 1.5 broke linear separability entirely. The boundary never settled. Exercise 3 stripped the implementation to blank `TODO` blocks: initialize weights and bias, write the forward pass, code the training loop. Three progressive hints moved from a conceptual sketch down to near-complete code with one line left open. Creative extensions closed each module. In M5 a challenge asked four perceptrons trained at different angles to split the canvas into sixteen coloured regions, turning weight-update arithmetic into a visual composition.

The theory-to-practice ratio in CDD exercises is approximately 50:50, double the theory weight of Hands-On Discovery (HOD). That extra reading is not padding. M6 (DDPM Basics) requires the learner to understand forward diffusion (adding Gaussian noise to an image over a sequence of time steps), a noise schedule (the variance β_t at each step), and reverse denoising (training a neural network to predict and remove the added noise). None of those mechanisms is visible in the output. A denoised image looks the same regardless of whether the noise schedule was linear, cosine, or sigmoid. The theory carries what the pixels cannot.

CDD exercises ran longer during the workshop. M3 (Convolution) took approximately 40 minutes, M5 (Perceptron) about 50, and M6 (DDPM Basics) about 40. The additional time

came almost entirely from the theory sections, not from additional coding. Participants read more, and the reading was denser: mathematical notation appeared in M5 (the perceptron update rule) and M6 (the noise schedule and the training-versus-sampling asymmetry). That density is unavoidable for those topics, but it carries a cost. Workshop data showed that CDD exercises produced significantly higher mental demand ratings than their Hands-On Discovery (HOD) counterparts ($W = 1.0$, p = .016, d = 1.100; full analysis in Chapter 5). Higher load was an accepted tradeoff, not a design failure. When the target audience lacks the prior knowledge to fill gaps left by minimal guidance, structured conceptual scaffolding is not optional (Kirschner et al. 2006). Conceptual Deep-Dive (CDD) pays the cognitive cost up front so that the learner has enough schema in place to make sense of the code when it arrives.

4.4. Assessment Design

Three instruments were designed or selected to match the curriculum's structure. Chapter 3 describes their administration procedures and scoring rules (Sections 3.6.1 to 3.6.3); what follows explains why each instrument takes the shape it does, and how the three fit together as an assessment layer on top of the six workshop modules.

The knowledge test comprised 24 multiple-choice items split into four sections of six items each. Each section mapped onto one or two adjacent workshop modules: Section A covered RGB arrays and cellular automata (M1–M2), Section B covered convolution and fractal recursion (M3–M4), Section C covered perceptron training (M5), and Section D covered diffusion models (M6). Items were written against the learning objectives of the corresponding modules, not against a generic computer-science syllabus. A question about kernel weights, for instance, tested exactly the concept that M3's Conceptual Deep-Dive (CDD) theory section had explained. That tight coupling between test content and module content is what makes pre-to-post gains interpretable as evidence of curriculum effect rather than general maturation. Each item offered four answer choices plus an "I Don't Know" option. Including that fifth choice was a deliberate design decision: it separated genuine ignorance from lucky guesses and produced a trackable marker of confidence boundaries. The drop in "I Don't Know" selections from pre-test to post-test turned out to be a finding in its own right (see Chapter 5).

Cognitive load measurement needed per-module resolution, not just a single end-of-day score. A curriculum built on progressive complexity makes a claim about how load changes across the sequence; testing that claim requires a repeated measure. The NASA Task Load Index (Hart et al. 1988) was selected for three reasons. It is one of the most widely validated workload instruments in human-factors research. Its six subscales (Mental Demand, Physical Demand, Temporal Demand, Performance, Effort, Frustration) separate different sources of load rather than collapsing them into a single number. And it is short enough to administer six times in one day without eating into the teaching schedule. The raw version was used, skipping the card-sorting weighting step from the original protocol. Running that procedure

after every module would have added roughly five minutes per round, thirty minutes total, time better spent on the curriculum itself. The tradeoff was accepted: raw TLX scores correlate highly with weighted scores in most applied settings (Hart 2006), and the six individual subscale profiles were more informative for this study's purposes than a single weighted composite would have been.

Exit tickets filled the gap that neither the knowledge test nor the TLX could reach. A correct answer on the post-test says nothing about *how* the learner arrived at understanding. A low TLX score says the module felt manageable but not *why*. Four open-ended prompts (What did you learn? How does this connect to earlier modules? What was most confusing? What would you like to explore next?) and one Likert-scale self-assessment of understanding (Q4, rated 1–5) went out after every module alongside the TLX form. The open prompts were short by design: one to three sentences per answer, collected on paper while the material was still fresh. That brevity limited the depth of any single response but ensured compliance; asking for a 500-word reflection after each of six modules would have produced fatigue, not insight.

Together, the three instruments formed a convergent assessment design. The knowledge test tracked what participants *knew* before and after the workshop. The TLX tracked how *hard* each module felt. The exit tickets tracked *why* participants found specific modules easy or difficult. When the quantitative instruments agreed with the qualitative responses, confidence in the finding rose; when they pulled apart, the mismatch pointed to something worth investigating. That convergence logic is the study's primary answer to RQ4, which asks how learning outcomes in creative AI education can be effectively assessed. No single instrument would have sufficed. The post-test KR-20 of 0.913 speaks to the reliability of the knowledge measure, but reliability alone does not cover the breadth of what this curriculum teaches. Procedural skill, conceptual understanding, and subjective difficulty occupy different assessment dimensions, and the instrument set was designed to cover all three.

4.5. Technology Stack and Platform

Python 3.11 was the sole programming language across all fifteen modules. The choice was pragmatic: Python dominates introductory programming courses, data science workflows, and machine learning research, so most learners in the target audience had at least passing familiarity with its syntax. Three libraries carried the bulk of the computation. NumPy handled array creation and element-wise arithmetic for every exercise from Module 1 onward. Pillow converted those arrays into displayable PNG and GIF files through its `Image.fromarray()` method. SciPy provided the `ndimage.convolve` function used in the cellular automata and convolution exercises, where a manual nested-loop implementation would have obscured the pedagogical point behind index arithmetic.

Modules in the upper half of the curriculum pulled in heavier dependencies. Matplotlib drew decision boundaries and training-loss curves for the perceptron exercise (M5). PyTorch

powered the neural network layers and the denoising diffusion model in Modules 9 and 12. PyTorch was chosen over TensorFlow for one reason: its eager-execution model prints intermediate tensor values without requiring a compilation step, a property that keeps every computation inspectable during a teaching session. Peripheral libraries filled narrower roles: OpenCV for geometric transforms in Module 3, the `noise` package for Perlin noise generation in Module 6, and Pygame for real-time rendering in later animation exercises. The full dependency list, pinned to minimum compatible versions, sits in a `requirements.txt` at the repository root.

The documentation platform (an HTML site built with Sphinx and the PyData theme, deployed to GitHub Pages) was described in the chapter introduction; the relevant design rationale for instructional layout and the two-window workflow appears there. On the deployment side, a GitHub Actions pipeline rebuilt the site on every push to the main branch: the workflow checked out the repository, installed the documentation dependencies from a separate `dev_requirements.txt`, compiled the reStructuredText and Markdown sources to static HTML, and uploaded the result to GitHub Pages. Build times sat under two minutes. That automation meant the curriculum site always reflected the latest commit, and any error in the documentation build surfaced immediately as a failed pipeline run rather than silently producing a broken page.

All source code and documentation sit in a public repository under an MIT license. Learners can clone the repository, run every script locally, and submit modifications through pull requests. Open access was a deliberate choice, not a default: the study evaluates a pedagogical approach, and locking the materials behind institutional access would have undercut both reproducibility and practical uptake.

4.6. Workshop Module Descriptions

Six exercises, drawn from six of the fifteen modules, formed the workshop tested on February 12, 2026. Three used Hands-On Discovery (HOD) and three used Conceptual Deep-Dive (CDD). Table 4.2 summarises the six modules with their framework assignments, durations, and opening hooks. What follows describes each one in the order participants encountered it.

Table 4.2.: Overview of the six workshop modules tested on February 12, 2026.

#	Module	Fw.	Dur.
M1	RGB Color Basics	HOD	∼30 min
M2	Cellular Automata	HOD	∼30 min
M3	Convolution & Kernels	CDD	∼40 min
M4	Fractal Square	HOD	∼30 min
M5	Perceptron from Scratch	CDD	∼50 min
M6	DDPM Basics	CDD	∼40 min

Note. HOD = Hands-On Discovery; CDD = Conceptual Deep-Dive. Each module followed a three-exercise scaffold (Execute, Modify, Create) plus optional extensions and creative challenges. Durations reflect observed workshop time, not the shorter estimates in the module metadata.

4.6.1. M1: RGB Color Basics (HOD)

The opening exercise taught images as numerical data. Participants created a 200×200 pixel canvas stored as a three-dimensional NumPy array with shape `(200, 200, 3)`, where the third axis held red, green, and blue channels. The Quick Start script filled the top half of the canvas with cyan and the bottom half with magenta by setting channel values to 0 or 255. That two-tone rectangle appeared on screen within the first two minutes, before any explanation of how RGB colour mixing works. Core Concepts covered three topics in sequence: grayscale images as two-dimensional arrays with brightness values from 0 (black) to 255 (white), the extension to three-channel colour through the `uint8` data type, and the additive colour model where red plus green light produces yellow rather than the brown that paint mixing would give. The distinction between additive and subtractive mixing was new to several participants and surfaced again in exit-ticket responses.

Exercise 1 (Execute). A provided script filled a 150×150 canvas with a single RGB triplet and asked participants to predict the colour before checking the output. The result was orange (255, 128, 0). Not an obvious guess for anyone unfamiliar with channel arithmetic.

Exercise 2 (Modify). Three colour-matching goals: produce pure cyan, medium grey, and dark purple by adjusting channel values alone. Cyan needed green and blue at maximum with red off; grey needed equal values across all three channels; dark purple called for red and blue at low intensity with green at zero. Each goal forced a different form of reasoning about how channels combine.

Exercise 3 (Create). Participants wrote a loop that painted a horizontal gradient from red on the left edge to blue on the right. Hints arrived in three tiers: the first explained proportional mapping from column index to intensity, the second identified which channels increase and which decrease, and the third provided partial code with one line left blank. “Make It Your Own” extensions invited vertical gradients, diagonal gradients, and a yellow-to-cyan transition that required working out which channels to swap.

Figure 4.1 shows the additive colour mixing output that participants produced.

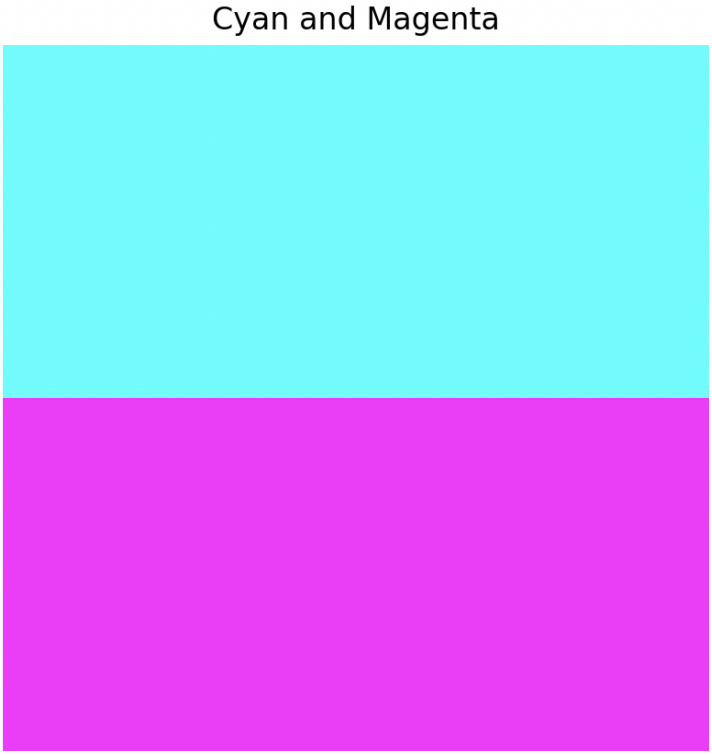


Figure 4.1.: RGB additive color mixing output from Module 1.

4.6.2. M2: Cellular Automata (HOD)

Where M1 treated each pixel independently, M2 introduced spatial relationships between cells. The exercise implemented Conway's Game of Life (Gardner 1970) on a 20×20 grid. A Quick Start script placed a glider pattern on the grid and ran eight time steps, saving the result as an animated GIF. Participants watched the five-cell glider crawl diagonally across the canvas before reading any rules. Core Concepts covered four topics: the simultaneity principle (all cells update from the current generation's state, not from partially updated values), Moore neighborhoods (the eight surrounding cells), the birth-survival-death rules (a dead cell with exactly three live neighbors becomes alive; a live cell with two or three neighbors survives; all others die), and boundary conditions. The boundary choice turned out to matter. Under wrap-around mode, patterns that left one edge reappeared on the opposite side, and the grid behaved as a torus. Fixed-edge mode treated cells beyond the boundary as permanently dead, starving edge-adjacent patterns of neighbors and killing them within a few generations. The implementation counted neighbors through `scipy.ndimage.convolve` with a 3×3 kernel of ones (center zeroed), replacing what would otherwise be eight separate index checks.

Exercise 1 (Execute). A blinker oscillator, a three-cell horizontal line that flipped to vertical and back every generation, ran on the grid. Participants tracked whether the living-cell count changed across frames. It does not. That observation separated oscillators from spaceships, which hold their cell count but shift position.

Exercise 2 (Modify). Three modification goals. Moving the blinker to the top edge had no effect under wrap-around mode, but switching to fixed edges killed it in two generations; the missing neighbor row broke the birth condition. A second goal placed multiple blinkers on the grid and let participants watch them oscillate independently when spaced apart, then interfere when placed two rows apart.

Exercise 3 (Create). A "Pattern Garden" combined a beacon oscillator (two 2×2 blocks offset diagonally, blinking between six and eight cells) with a glider spaceship on the same grid. Participants wrote the evolution loop, printed the generation number and living-cell count at each step, and watched the beacon pulse in place while the glider drifted diagonally across twenty generations. A challenge extension placed two gliders on a collision course and ran fifty or more generations to observe whether they annihilated, merged into a still life, or produced new moving structures.

Figure 4.2 shows three canonical Game of Life patterns that participants encountered: a period-2 blinker oscillator, a beacon oscillator, and a glider spaceship.

(a) Blinker (period-2) (b) Beacon (period-2) (c) Glider (spaceship)

Figure 4.2.: Canonical Game of Life patterns from the Cellular Automata module (Module 2). Each panel shows a single frame from the animated output.

4.6.3. M3: Convolution and Kernels (CDD)

M3 was the first Conceptual Deep-Dive (CDD) exercise in the workshop sequence. The opening hook framed convolution as a sliding operation: a small grid of numbers moves across every pixel, and the weighted sum at each position transforms the image. The Quick Start made that concrete with a 256×256 checkerboard and a 5×5 blur kernel whose twenty-five cells each held $1/25$. Two nested loops walked the kernel across the grid, multiplying each overlapping patch and summing the products into one output value. Sharp black-white edges turned to gray gradients. The mechanism was visible before any formal vocabulary appeared.

Three Core Concept sections followed. The first explained the three-step logic of convolution: position the kernel, multiply element-wise, sum. The second covered four kernel types.

An identity kernel (center one, rest zero) passed the image through unchanged, serving as a no-operation baseline. Blur worked the opposite way: all values equal, normalized so they sum to one, averaging neighbors into soft gradients. Sharpening set the center to five and each cross-neighbor to negative one; the weights still sum to one, so overall brightness holds while edges grow crisper. Edge detection used a Laplacian layout: center eight, all eight neighbors negative one. That kernel sums to zero. Uniform regions go black; only intensity boundaries light up. The third Core Concept addressed borders. When the kernel slides past the image edge, the missing pixels have to come from somewhere. Zero-padding fills them with black, which can darken borders. Edge-replication copies the nearest row or column outward. Reflection mirrors the boundary region.

Exercise 1 (Execute). All four kernel types applied to the Brandenburg Gate in a single run, producing a two-by-two comparison grid. Reflection questions asked why uniform sky went black under edge detection and why sharpened text looked crisper.

Exercise 2 (Modify). Four modification goals: double brightness by changing the center weight to two, blur the image by setting all nine values to one-ninth, sharpen with the five-and-negative-one pattern, detect edges with the Laplacian. Normalization tripped people up. Forgetting to divide the blur kernel by nine doubled or tripled overall brightness, a mistake that made the principle stick.

Exercise 3 (Create). The convolution stripped to four `TODO` blocks. Participants wrote the row loop, the column loop, the region extraction (`padded[y:y+k, x:x+k]`), and the weighted sum (`np.sum(region * kernel)`). Four lines of code total, but getting the index arithmetic right forced participants to think through how padding width, kernel size, and output dimensions connect. Hints followed the same three-tier scaffold used across the curriculum.

An accompanying Jupyter notebook, `GenerativeConvolution.ipynb`, offered interactive sliders for kernel weights; participants who finished early could watch blur strength or edge response shift in real time. The nested-loop implementation accounted for most of M3's time: reasoning about index offsets is slower than calling a library function, but it is also harder to forget. Figure 4.3 shows the kernel effects that participants observed.

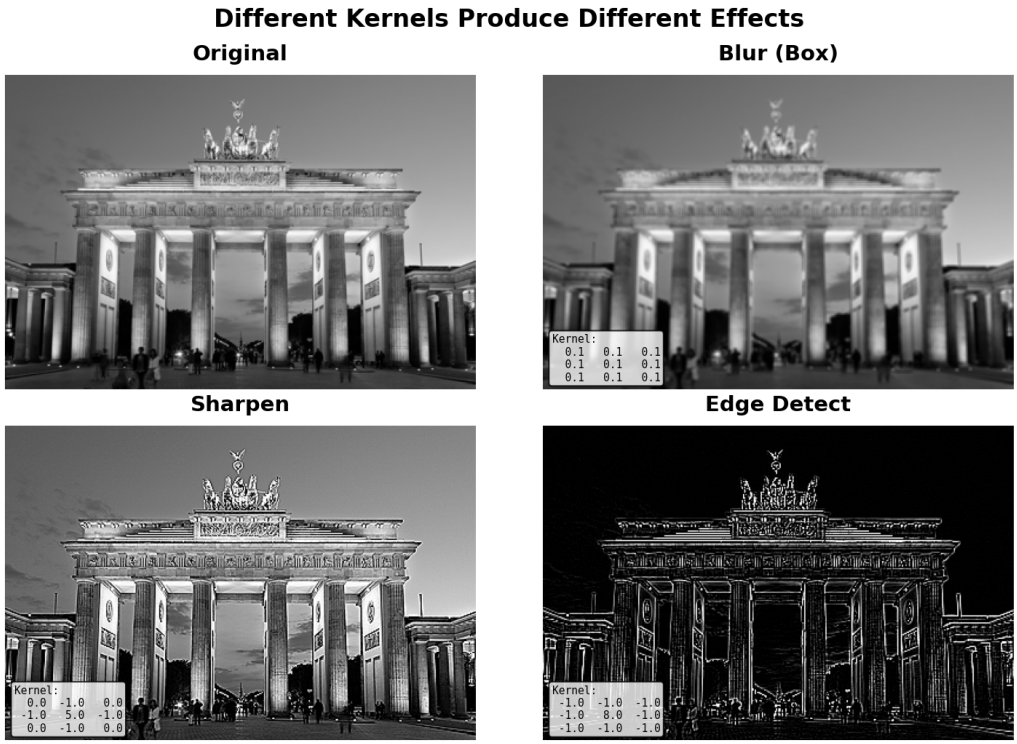


Figure 4.3.: Convolution kernel effects from Module 3. Source photograph: Brandenburg Gate (Wolf 2012).

4.6.4. M4: Fractal Square (HOD)

The fractal exercise applied Hands-On Discovery (HOD) to recursion, a topic usually taught through abstract numerical examples. The Quick Start displayed an 800×800 black canvas covered in nested green squares at four levels of detail. Participants executed the script and watched the fractal fill the screen before the word “recursion” appeared in any explanation. Core Concepts broke the algorithm into three steps. First, the current region divides into a 3×3 grid using integer division. Second, the center cell fills with a colour increment applied through the += operator, so overlapping regions accumulate brightness rather than overwriting it. Third, the function calls itself on the four corner sub-regions, each covering a $\frac{2}{3} \times \frac{2}{3}$ block that overlaps with the center. Overlap is the mechanism behind the nested glow visible in the output: squares drawn at different depths stack their colour values. At depth n the algorithm produces $(4^{n+1} - 1)/3$ center squares, growing from 1 at depth 0 to 1,365 at depth 5.

Four configuration parameters controlled the output: `RECURSION_DEPTH` (levels of detail, 0 through 5), `COLOR_CHANNEL` (0 for red, 1 for green, 2 for blue), `COLOR_INCREMENT` (brightness added per level; 16, 32, or 64), and `CANVAS_SIZE` (pixel dimensions). Setting the increment too high risked integer overflow past the `uint8` ceiling of 255, which clipped brightness and flattened the gradient. Participants discovered that constraint by experiment, not by being told.

Exercise 1 (Execute). The fractal ran at depth 3. Three reflection questions followed: how many distinct brightness levels are visible (four, one per recursion level), why certain areas

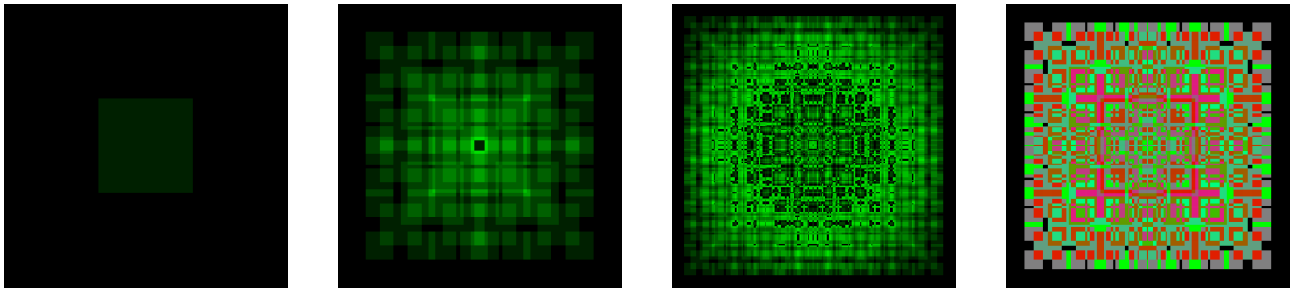
appear brighter (overlapping recursive regions accumulate colour), and what changes when depth drops to zero (a single centered square) or rises to five (a dense lattice with 1,365 cells).

Exercise 2 (Modify). Three parameter-modification goals. Changing depth from 0 to 5 traced the exponential growth of recursive calls. Switching the colour channel from green to red or blue produced structurally identical fractals in a different hue, isolating colour from geometry. Raising the increment to 64 produced high-contrast output but pushed deep regions past 255, clipping them to white.

Exercise 3 (Create). The recursive calls stripped to four blank `TODO` blocks. Participants wrote each corner call using the coordinate variables defined in the setup code. The hint scaffold progressed from a spatial sketch of which cells the four corners occupy, through one completed corner call as a pattern, to three of the four calls with the last left open. "Make It Your Own" extensions invited depth-5 runs, colour channel substitution, and removing two of the four corner calls to generate asymmetric compositions.

Creative Challenge. Participants mapped recursion depth to RGB values so that shallow squares carried one hue and deep squares another, turning the monochrome fractal into a gradient composition.

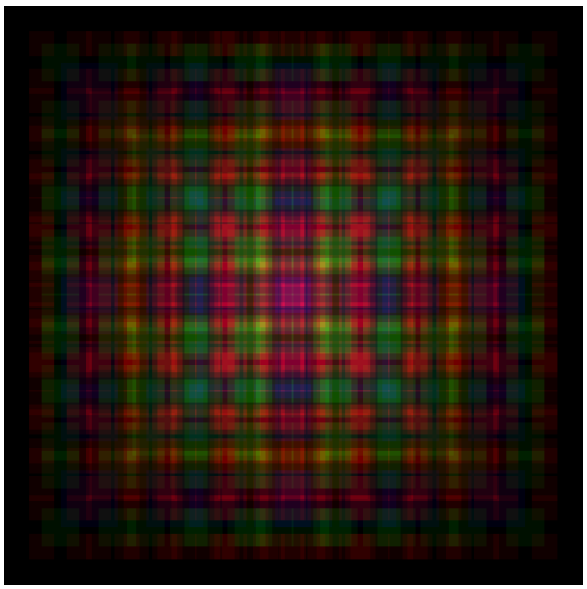
Figure 4.4 traces how recursion depth shapes the visual output, from the depth-0 base case through to the complete fractal lattice and a colour-mapped challenge extension.



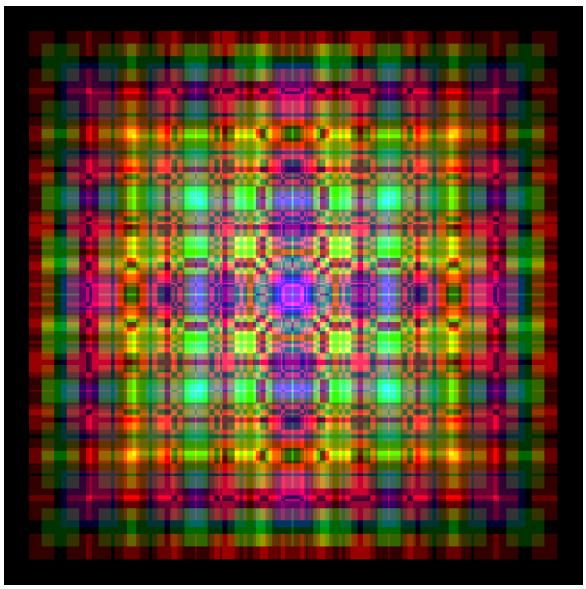
(a) Depth 0 (b) Depth 3 (c) Depth 5 (d) Colour challenge

Figure 4.4.: Fractal Squares outputs at increasing recursion depths (Module 4). Sub-figures (a)–(c) show the canonical progression; (d) applies depth-based colour mapping.

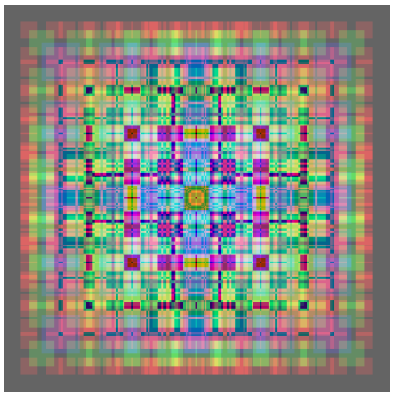
Workshop participants pushed these parameters in varied directions. Exercise 2 outputs ranged from a high-contrast rainbow grid to a softly glowing radial pattern, both produced by adjusting the `COLOR_CHANNEL` and `COLOR_INCREMENT` configuration values. In Exercise 3, after writing the recursive corner calls from scratch, participants applied the module's "Make It Your Own" extensions: one substituted the colour palette and background tone, another ran the recursion to depth 5 with multi-channel colour mapping, and a third removed two of the four corner calls to generate the asymmetric diagonal composition in Figure 4.5(e), following the module's explicit structural extension prompt.



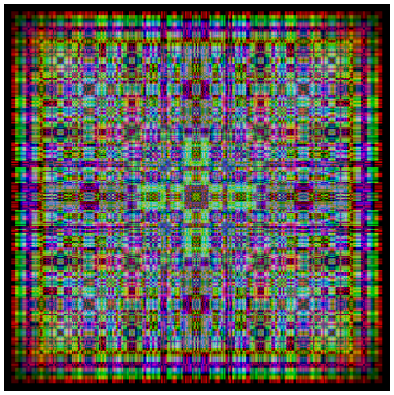
(a) Ex. 2: soft radial glow



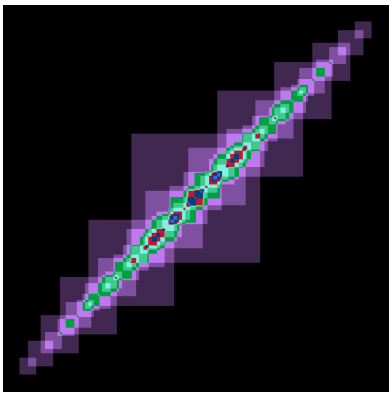
(b) Ex. 2: high-contrast rainbow



(c) Ex. 3: palette substitution



(d) Ex. 3: depth 5, multi-channel



(e) Ex. 3: asymmetric diagonal

Figure 4.5.: Participant-generated outputs from the Fractal Squares module. (a)–(b) Exercise 2 parameter variations on the canonical structure; (c)–(e) Exercise 3 outputs applying the “Make It Your Own” extensions.

4.6.5. M5: Perceptron from Scratch (CDD)

M5 was the most time-intensive exercise in the workshop and the first to enter neural network territory. The opening hook traced the perceptron back to Rosenblatt’s 1958 architecture: multiply inputs by weights, sum the products, check whether the result crosses a threshold. Three Core Concept sections unpacked that sequence. The first covered the architecture itself, formalizing the step activation function (output $= 1$ if $\mathbf{w} \cdot \mathbf{x} + b \geq 0$, else 0) and explaining how weights control the slope of the decision boundary while the bias shifts its position. The second section walked through a forward-pass implementation built on `np.dot` and a threshold check. Core Concept 3 introduced the learning rule, $w \leftarrow w + \eta(y - \hat{y})x$, with a worked code block that ran weight updates across several epochs on sample data. All three sections included reference code that the learner read but did not yet modify; the hands-on implementation came in the exercises that followed.

Exercise 1 (Execute). A pre-built perceptron ran on generated data. Reflection questions asked how many epochs until zero errors, and what relationship between weights and bias fixes the boundary angle.

Exercise 2 (Modify). Three parameter-modification goals that pushed the algorithm past its limits. At a learning rate of 0.01, convergence slowed visibly compared to the default 0.1. Widening cluster spread to 1.5 made the data non-linearly separable; the perceptron never settled, looping through weight configurations without finding a valid boundary. That failure is not a bug. It is the XOR limitation, the same constraint that Minsky and Papert formalized (Minsky et al. 1969) and that cut neural network funding for over a decade. A third goal pushed cluster distance down to 0.3, where the boundary swung back and forth as each correction undid the previous one.

Exercise 3 (Create). Three blank `TODO` blocks: initialize weights and bias, write the forward pass (dot product plus step function), and code the update rule inside a training loop. Hints arrived in three tiers, from a conceptual sketch of the algorithm down to near-complete code with one line left open.

The module embedded this algorithm in its historical context, tracing from Rosenblatt's 1958 perceptron (Rosenblatt 1958) through the Minsky and Papert proof (Minsky et al. 1969) to the funding collapse that followed. Exercise 2's non-separable failure case carried more weight for participants who had just read that history. A creative challenge closed the module. Four perceptrons, each trained at a different angle, fed their binary outputs into a four-bit signature that split the canvas into sixteen coloured regions. The output was an animated GIF: geometric boundaries shifted and locked into place as training converged, turning weight-update arithmetic into a visual composition. Mathematical notation throughout (the update rule, the dot product, the activation threshold) sat well above the level of earlier exercises, and the workshop load data placed M5 among the highest mental demand ratings in the sequence (Chapter 5). Figure 4.6 shows the decision boundary that participants produced.

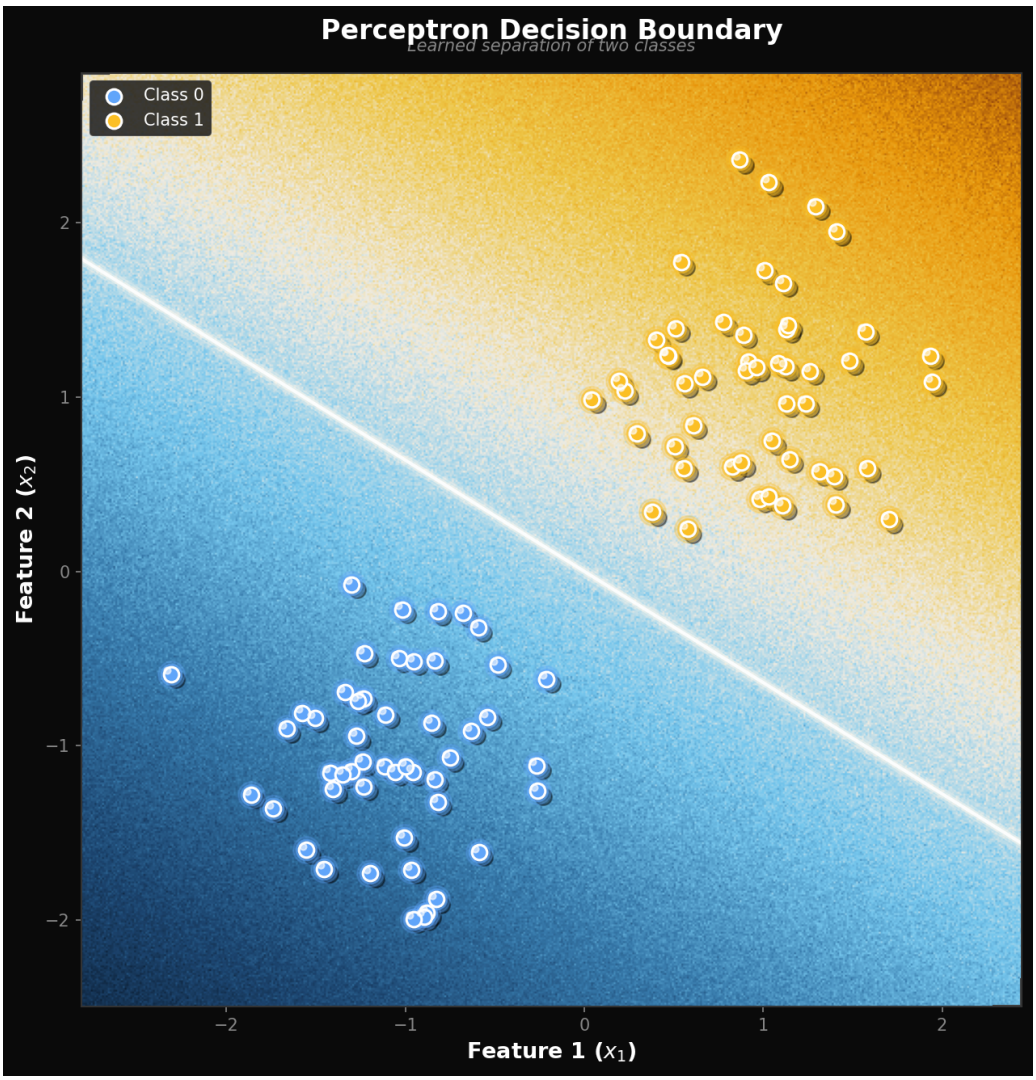


Figure 4.6.: Perceptron decision boundary output from Module 5.

4.6.6. M6: DDPM Basics (CDD)

The final exercise sat at the curriculum's most advanced point. The opening hook stated that diffusion models displaced GANs as the dominant generative architecture within two years of their introduction, and the module set out to explain why. Four Core Concept sections built the theoretical base. The first framed the diffusion approach against GANs: stable MSE training instead of adversarial dynamics, no mode collapse, competitive image quality. The second covered forward diffusion. A clean image was corrupted over a sequence of time steps by adding Gaussian noise at each step until only static remained. The module compared linear and cosine noise schedules side by side; under the cosine variant, recognizable structure survived further into the corruption sequence, giving the network more informative training examples at middle time steps. Core Concept 3 explained the reverse process. A U-Net predicted the noise present at each step so that running the chain backward recovered a structured image from pure static. Four modifications separated this U-Net from a plain encoder-decoder: sinusoidal timestep embeddings encoded the current noise level as a high-dimensional vector, self-attention layers let distant spatial positions exchange information

across the feature map, residual connections fed each block's input directly to its output, and group normalization replaced batch normalization to stabilize training at small batch sizes. The training objective was straightforward in principle (minimize the L2 distance between predicted and actual noise) but rested on concepts, variance schedules, reparameterization, skip connections, that none of the earlier modules had introduced. A fourth Core Concept section covered the asymmetry between training and sampling: training processes random timesteps independently, while sampling must walk the full chain sequentially from $t = T$ to $t = 0$. DDIM, a deterministic reformulation, cut that walk from 1,000 steps to 250 with negligible quality loss.

Exercise 1 (Execute). Pre-trained weights loaded and sixteen fabric patterns generated on CPU. Generation took roughly thirty seconds because the module used DDIM, a deterministic reformulation that cut the 1,000 training time steps down to 250 at sampling time with negligible quality loss. Participants compared those outputs against DCGAN results from an earlier curriculum module, both trained on the same 1,059 African fabric images. Same data, different architecture; the visual gap was immediate.

Exercise 2 (Modify). Four parameter goals. The first varied the number of DDIM steps to expose the speed-quality trade-off. The second stepped through early forward-diffusion stages, where the first twenty-five steps added almost imperceptible grain. The third changed the random seed, confirming that each starting point locks in a distinct pattern. The fourth captured intermediate denoising frames, showing that large-scale colour regions resolve well before fine texture appears.

Exercise 3 (Create). Participants built a morphing animation. Two noise vectors, each seeded to produce a different fabric pattern, were blended through linear interpolation at twenty evenly spaced ratios. Each blended vector was denoised into a full image, and the frames were stitched into a GIF that walked smoothly from one textile motif to another. A challenge extension replaced linear blending with spherical interpolation, which held vector magnitude constant and produced sharper mid-sequence frames.

Two paths through the training component were available. Option A used the pre-trained weights and ran on CPU, keeping the workshop on schedule. Option B trained a DDPM from scratch: 500,000 optimization steps, roughly 15–20 hours on a consumer GPU, available for participants to attempt after the session but not during it. All nine took Option A. The generated fabric patterns, emerging from random noise into recognizable textile motifs, closed the curriculum arc that had begun with single-pixel colour assignment in M1. Figure 4.7 shows the forward diffusion process that participants observed.

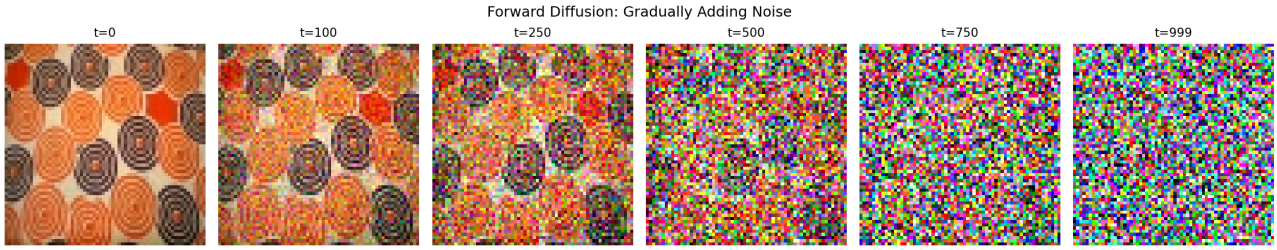


Figure 4.7.: Forward diffusion process visualization from Module 6.

5. Results

Eight paired pre-post comparisons, 54 NASA-TLX module ratings, and 47 exit-ticket response sets form the dataset reported in this chapter. One participant (P04) completed all modules and TLX surveys but did not submit exit tickets from M2 onward or the post-test, reducing paired knowledge analyses to $n = 8$; cognitive load data are complete for all nine participants across all six modules. Data were collected through four primary instruments:

1. a 24-question multiple-choice knowledge test administered before and after the workshop,
2. the NASA Task Load Index (Hart et al. 1988) completed after each of the six modules,
3. exit tickets combining four open-ended questions with one Likert-scale item per module, and
4. a demographics questionnaire.

A brief post-workshop evaluation form was also distributed at session close; responses from seven of nine participants are reported descriptively as supplementary evidence in Section 5.7. Exit ticket responses underwent reflexive thematic analysis following the six-phase procedure outlined in Chapter 3. Interpretation against the research questions follows in Chapter 6.

5.1. Participant Demographics and Data Completeness

Nine participants, coded P01 through P09, attended the workshop. The sample split five male, four female. Ages spanned three brackets: three participants aged 18–24, four aged 25–34, and two aged 35–44. Four held master's degrees, two had bachelor's degrees, and three had completed high school as their highest qualification. Table 5.1 summarizes the full demographic profile.

Programming backgrounds spanned a wide range, from near-zero to over a decade. Four participants reported less than one year of experience (44%), two reported one to two years (22%), two reported three to five years (22%), and one reported more than ten years (11%). Python proficiency followed a similar spread: two participants had no prior exposure, four rated themselves as beginners, two as intermediate, and one as advanced. A composite experience score, calculated on a 0–1 scale, put the sample mean at $M = 0.209$ ($SD = 0.162$), squarely in beginner territory, though one participant's score sat well above the rest.

Of 135 possible data cells across all instruments, 127 were filled, a completeness rate of 94.1%. One participant (P04) completed all six modules and all six TLX surveys but did not submit exit tickets from M2 onward and did not complete the post-test. A second participant (P06) was missing exit ticket data for Modules 5 and 6. The NASA-TLX dataset had no gaps: every participant rated every module.

P04's missing post-test reduces all paired pre–post comparisons to $n = 8$. Cognitive load analyses retain the full sample of nine. Exit ticket qualitative analyses have a variable respondent count per module, ranging from seven to nine. One participant (P08) was flagged during data validation for a possible satisficing pattern, having rated four of six TLX subscales uniformly at the minimum value across all modules. Excluding P08 from sensitivity analyses did not change any of the key results reported below.

Table 5.1.: Participant Demographics ($N = 9$)

Characteristic	**Category**	***n***	**%**
Gender	Male	5	55.6
	Female	4	44.4
Age range	18–24	3	33.3
	25–34	4	44.4
	35–44	2	22.2
Education	High school	3	33.3
	Bachelor's degree	2	22.2
	Master's degree	4	44.4
Programming	Less than 1 year	4	44.4
experience	1–2 years	2	22.2
	3–5 years	2	22.2
	More than 10 years	1	11.1
Python	No experience	2	22.2
proficiency	Beginner	4	44.4
	Intermediate	2	22.2
	Advanced	1	11.1
NumPy	Never used	4	44.4
proficiency	Basic awareness	3	33.3
	Beginner	2	22.2
Domain experience		***M***	***SD***
(1–5 scale)	Image processing	2.1	0.9
	Creative coding	1.9	1.2
	Machine learning	2.0	1.3
	Neural networks	1.8	1.1
	Generative AI	1.3	0.5
	TouchDesigner	1.3	1.0
Composite score	(0–1 scale, 8 items)	0.209	0.162

5.2. Knowledge Assessment Results

Pre-test scores averaged 3.62 out of 24 ($SD = 2.72$, $Mdn = 3.0$). By the post-test, that figure had tripled to 12.12 ($SD = 6.29$, $Mdn = 10.5$), a raw gain of 8.50 points or 35.4 percentage points. All eight paired participants improved their scores. Table 5.2 presents the full descriptive statistics and Figure 5.1 displays individual score trajectories.

Table 5.2.: Pre-Test and Post-Test Descriptive Statistics by Content Section ($n = 8$)

Section	Test	*M*	*Mdn*	*SD*	Min	Max	95% CI	*M*%
Overall (0–24)	Pre	3.62	3.0	2.72	1	9	[2.00, 5.50]	15.1
	Post	12.12	10.5	6.29	6	24	[8.38, 16.50]	50.5
A: Arrays & Images (0–6)	Pre	1.25	1.0	1.04	0	3	[0.62, 1.88]	20.8
	Post	3.62	3.0	1.51	2	6	[2.62, 4.62]	60.4
B: Computational (0–6)	Pre	0.88	0.0	1.46	0	4	[0.00, 1.88]	14.6
	Post	3.12	3.0	1.46	1	6	[2.25, 4.12]	52.1
C: Neural Networks (0–6)	Pre	1.00	1.0	1.07	0	3	[0.38, 1.75]	16.7
	Post	3.50	3.5	1.85	1	6	[2.25, 4.62]	58.3
D: Generative AI (0–6)	Pre	0.50	0.0	1.07	0	3	[0.00, 1.25]	8.3
	Post	1.88	1.0	2.30	0	6	[0.50, 3.50]	31.2

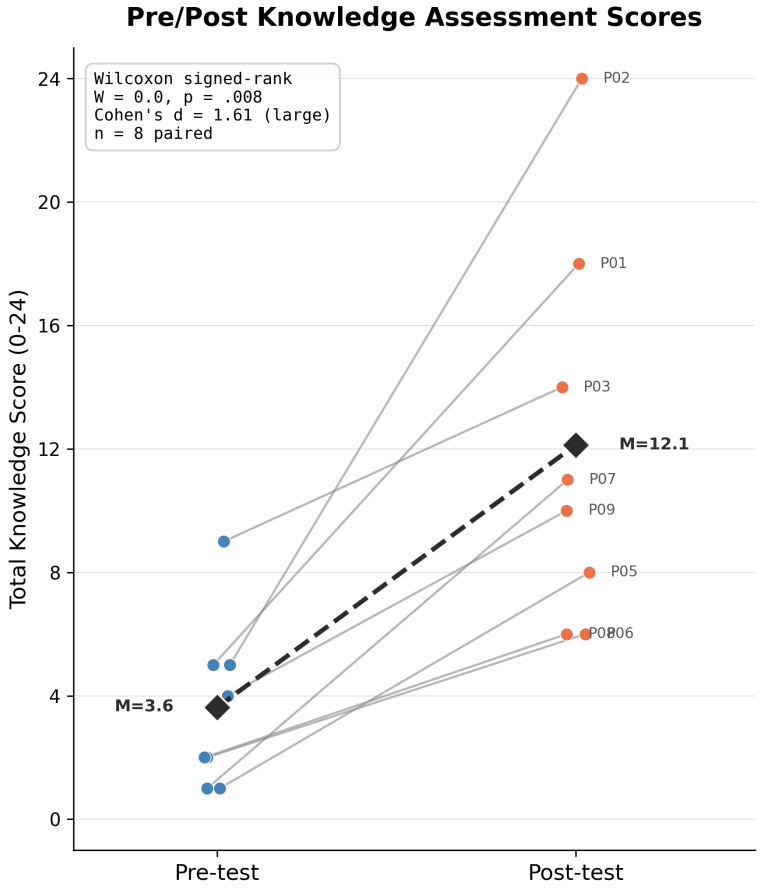


Figure 5.1.: Pre-test and post-test knowledge assessment scores ($n = 8$ paired). Gray lines connect individual trajectories; black diamonds indicate group means.

A Wilcoxon signed-rank test returned $W = 0.0$, $p = .008$ (exact), $d = 1.615$, $r_{rb} = 1.000$. All eight participants improved, with no negative ranks or ties. With $n = 8$ and zero negative ranks, this is the ceiling result; the test cannot produce a stronger outcome at this sample size.

Three of four content domains reached significance at the section level. Section A (Arrays and Images) produced the largest effect, $W = 0.0$, $p = .008$, $d = 1.687$, followed by Section C (Neural Networks), $W = 0.0$, $p = .031$, $d = 1.479$, and Section B (Computational Concepts), $W = 1.5$, $p = .023$, $d = 1.135$. Section D (Generative AI) was the only domain that did not reach significance, $W = 0.0$, $p = .062$, $d = 0.689$, with a residual "I don't know" rate of 56% on the post-test. Full test statistics appear in Table 5.3; Figure 5.2 plots the gains by domain.

Table 5.3.: Wilcoxon Signed-Rank Tests and Effect Sizes by Content Section ($n = 8$)

Section	Gain *M*	Gain *SD*	*W*	*p*	Sig.	*d*	Magnitude	r_{rb}
Overall (0–24)	8.50	5.26	0.0	.008	**	1.615	Large	1.000
A: Arrays & Images (0–6)	2.38	1.41	0.0	.008	**	1.687	Large	1.000
B: Computational (0–6)	2.25	1.98	1.5	.023	*	1.135	Large	0.917
C: Neural Networks (0–6)	2.50	1.69	0.0	.031	*	1.479	Large	1.000
D: Generative AI (0–6)	1.38	2.00	0.0	.062	ns	0.689	Medium	1.000

Note. W = Wilcoxon signed-rank statistic; p = exact p-value; d = Cohen's d; r_{rb} = rank-biserial correlation. *$p < .05$, **$p < .01$.

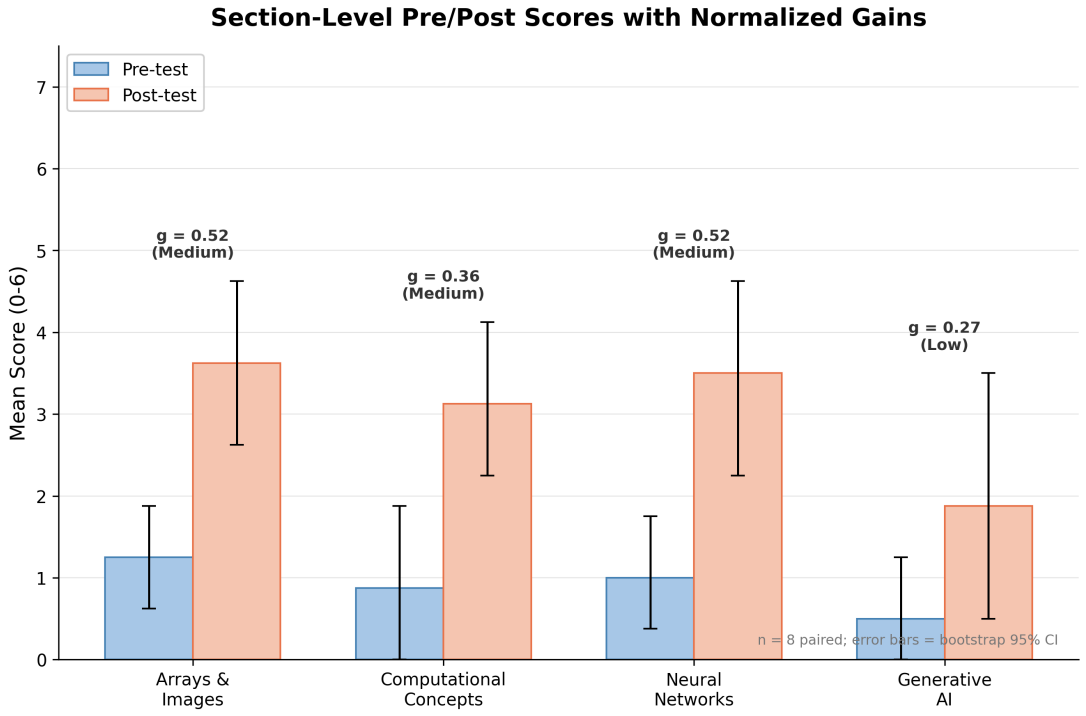


Figure 5.2.: Mean pre-test and post-test scores by content section with Hake normalized gains (g). Error bars represent bootstrap 95% confidence intervals ($n = 8$ paired).

Normalized learning gains, calculated using the Hake formula, (Hake 1998) produced an overall value of $g = 0.428$ (medium gain, 95% CI $[0.27, 0.63]$). Section-level gains ranged from

$g = 0.271$ (low) for Generative AI to $g = 0.517$ (medium) for both Arrays and Images and Neural Networks. P02 was the standout case, jumping from 5 to a perfect 24, a Hake gain of $g = 1.00$. Five participants landed in the medium-gain band; two (P06, P08) fell in the low range. Figure 5.3 displays the individual gain distribution.

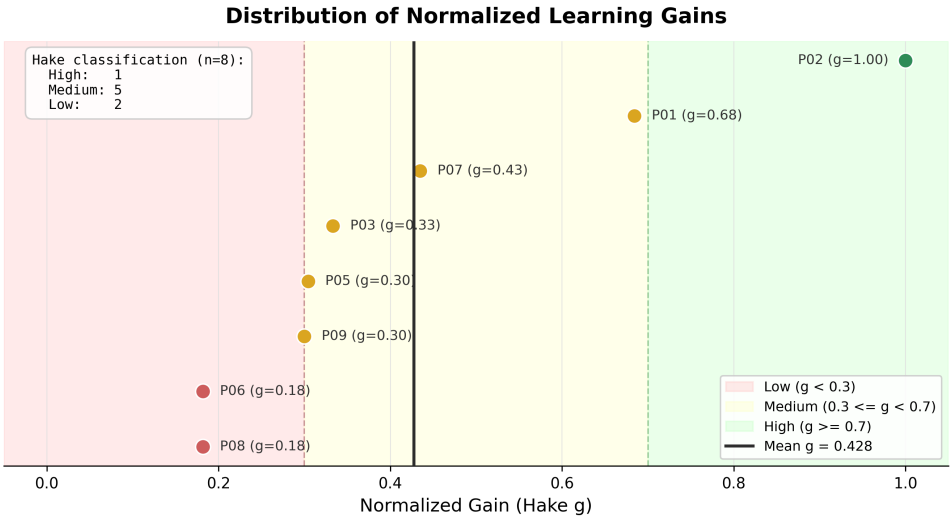


Figure 5.3.: Distribution of individual Hake normalized gains ($n = 8$). Shaded zones indicate gain classification (Hake 1998); dashed line marks the group mean.

The proportion of "I don't know" responses dropped from 74% on the pre-test to 24% on the post-test ($n = 8$ paired), a reduction of 50 percentage points. Sections A and B saw the steepest drop, with residual IDK rates near 10%. Section D still sat at 56% IDK on the post-test, the highest residual rate by a wide margin.

5.3. Cognitive Load Analysis

Module 1 (RGB Basics) came in at the lowest overall workload, $M = 7.74$ ($SD = 2.03$) on the NASA-TLX 1–20 scale. From there, scores jumped more than three points to Module 2 (Cellular Automata, $M = 10.94$, $SD = 3.70$), the peak of the entire six-module sequence. Module 3 (Convolution, $M = 9.93$) pulled back slightly, Module 4 (Fractal Square, $M = 8.59$) dipped further, and Modules 5 and 6 settled near the top again at $M = 10.70$ and $M = 10.67$, respectively. The pattern was sawtooth-like, not a steady climb. And the single highest value belonged to Module 2, which was an HOD module, not one of the three CDD sessions. Table 5.4 gives the descriptive statistics; Figure 5.4 plots the trajectory.

Table 5.4.: NASA-TLX Overall Workload by Module ($n = 9$, Scale 1–20)

Module	Framework	*M*	*Mdn*	*SD*	Min	Max
M1 RGB Basics	HOD	7.74	8.00	2.03	4.17	10.00
M2 Cellular Automata	HOD	10.94	11.00	3.70	5.83	15.17
M3 Convolution	CDD	9.93	10.83	3.34	4.83	15.17
M4 Fractal Square	HOD	8.59	8.83	2.57	3.50	12.83
M5 Perceptron	CDD	10.70	12.83	3.70	3.67	14.50
M6 DDPM Basics	CDD	10.67	11.83	3.11	4.67	13.67

Note. HOD = Hands-On Discovery; CDD = Conceptual Deep-Dive. Overall workload is the unweighted mean of six subscales.

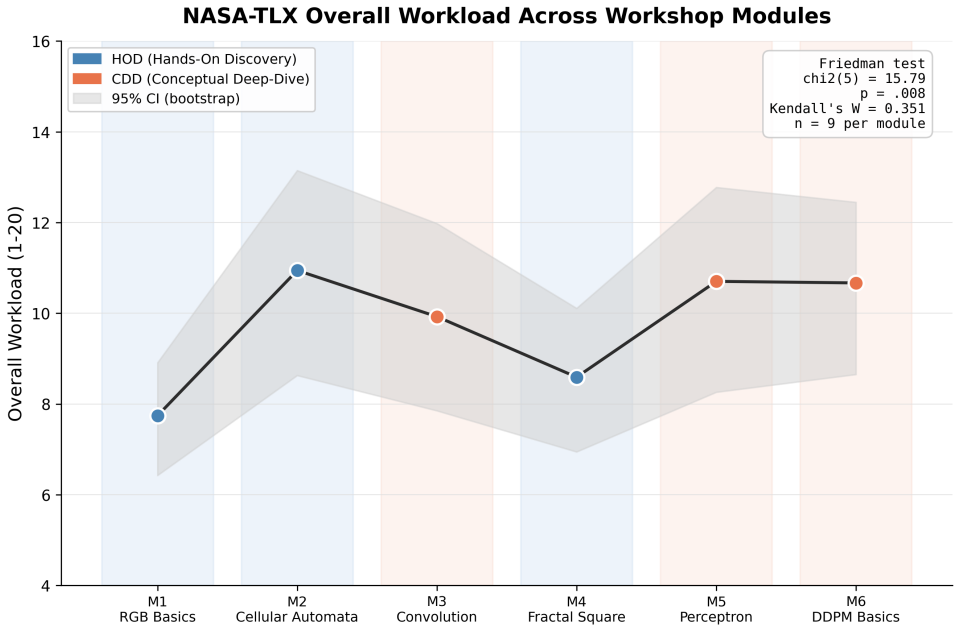


Figure 5.4.: NASA-TLX overall workload trajectory across six modules ($n = 9$). Shaded background indicates framework (blue = HOD, red = CDD); gray band shows bootstrap 95% confidence interval.

The Friedman test put a number on this variation: $\chi^2(5) = 15.795$, $p = .008$, Kendall's $W = 0.351$. That is a moderate effect. The NASA-TLX had no gaps in this study; all nine participants rated all six modules, a complete 9×6 matrix.

Table 5.5.: Friedman Test Results for NASA-TLX Subscales Across Six Modules ($n = 9$)

Subscale	$\chi^2(5)$	*p*	Sig.	Kendall's *W*
Overall Workload	15.795	.008	**	0.351
Mental Demand	24.827	<.001	***	0.552
Temporal Demand	18.618	.002	**	0.414
Effort	13.870	.017	*	0.308
Physical Demand	9.663	.085	ns	0.215
Performance (inv)	9.044	.107	ns	0.201
Frustration	9.385	.095	ns	0.209

Note. W = Kendall's coefficient of concordance. Performance is inverted so that higher values = greater perceived success. $^{*}p < .05$, $^{**}p < .01$, $^{***}p < .001$.

Four of the seven Friedman tests in Table 5.5 crossed the significance threshold. Mental Demand stood out, $\chi^2(5) = 24.827$, $p < .001$, $W = 0.552$, the only subscale whose Kendall's W topped 0.50. Temporal Demand ($\chi^2(5) = 18.618$, $p = .002$, $W = 0.414$) and Effort ($\chi^2(5) = 13.870$, $p = .017$, $W = 0.308$) were also significant, though with weaker concordance. Physical Demand ($p = .085$), Performance ($p = .107$), and Frustration ($p = .095$) all missed the $\alpha = .05$ cutoff. The spread of W values, from 0.201 at the low end to 0.552 at the high, puts the significant subscales in a moderate-to-strong band of agreement about which modules were harder.

All three significant subscales (Mental Demand, Temporal Demand, Effort) shared a basic shape: low at Module 1, higher everywhere else. But within that shared shape, the details diverged. Mental Demand was where the CDD modules separated most cleanly. The three CDD sessions (M3, M5, M6) all scored above 10, with M5 peaking at $M = 14.00$ ($SD = 6.04$). M1, the opening HOD module, came in at just 6.33. That is a gap of nearly eight points on a twenty-point scale. M2 broke the pattern. Despite being an HOD module, its Mental Demand ($M = 13.44$) matched M6 exactly, making it an outlier among the hands-on sessions. M4, at 9.44, was closer to M3's CDD value of 10.78 than to M1. Temporal Demand told a different story. Its two peaks were M2 ($M = 13.00$) and M6 ($M = 13.22$), while M4 bottomed out at $M = 6.56$. Nearly seven points separated top from bottom. Unlike Mental Demand, the Temporal Demand peaks did not split along framework lines; both an HOD module (M2) and a CDD module (M6) hit the ceiling. Effort peaked at M2 ($M = 12.67$) and M5 ($M = 12.78$) but had the weakest concordance of the three ($W = 0.308$). Standard deviations on all three subscales were at least 3.40 for every module, so person-to-person responses varied widely.

Physical Demand, Performance, and Frustration did not reach significance in the Friedman tests. Physical Demand never topped $M = 6.00$; it was the quietest subscale throughout. The more notable trends were in Performance and Frustration. Performance, scored so that higher values mean the participant felt more successful, held between 15 and 17 across the first four modules, then slid to $M = 13.11$ at M5 and $M = 11.56$ at M6. That M4-to-M6 drop of 5.44 points was the steepest single-subscale decline in the dataset. Frustration moved the other way, climbing from $M = 5.56$ at M1 to $M = 9.44$ at M6. Neither passed its Friedman threshold ($p = .095$ and $p = .107$), but together they paint a consistent picture of the final two modules: participants worked harder, felt less successful, and grew more frustrated.

Figure 5.5 breaks this down module by module. M1 and M6 look nothing alike. At M1, only Effort ($M = 8.56$) and Temporal Demand ($M = 7.22$) cleared 7.0; everything else hugged the floor. At M6, four subscales topped $M = 9.0$: Mental Demand at 13.44, Temporal Demand at 13.22, Effort at 11.78, Frustration at 9.44, while Performance sank to 11.56. Module 6 was not just harder on one dimension. It was harder across the board.

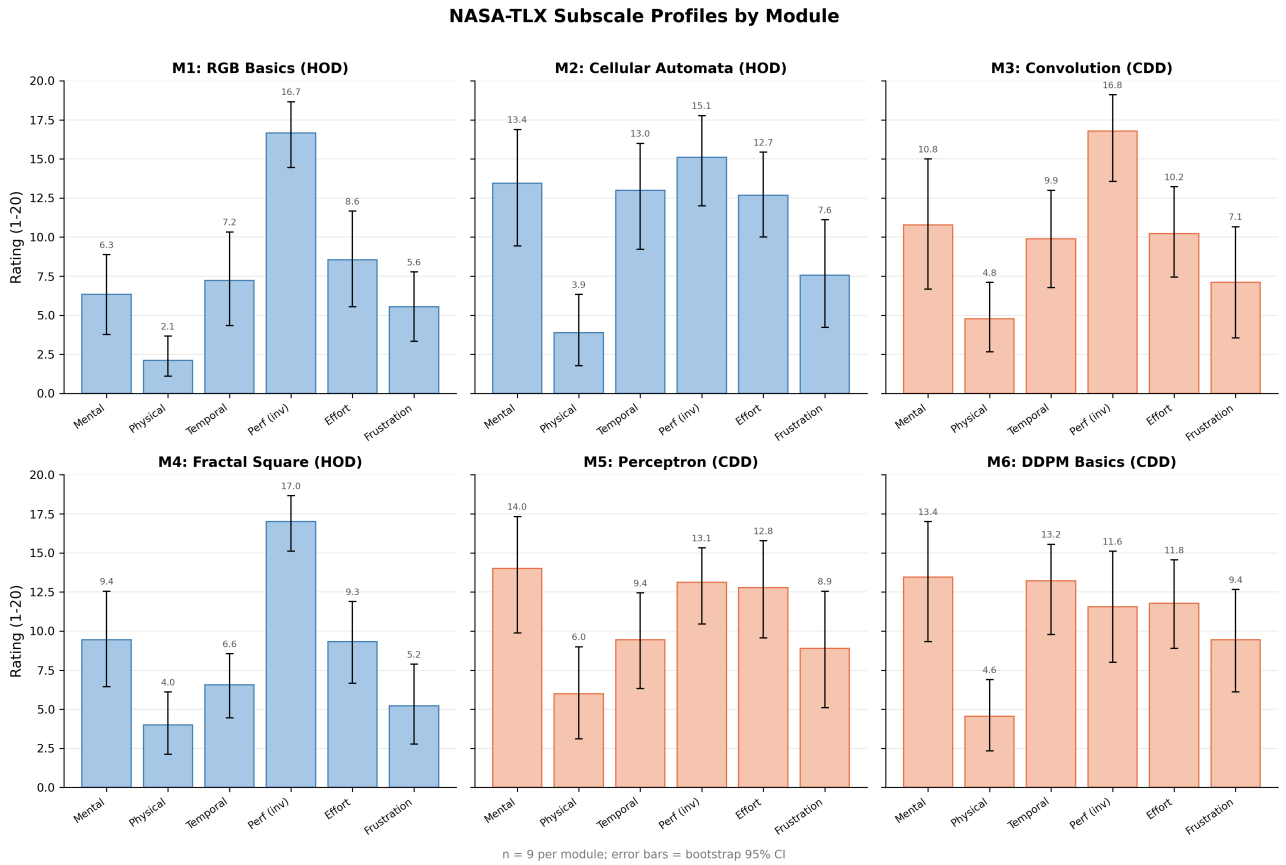


Figure 5.5.: NASA-TLX subscale profiles by module ($n = 9$, scale 1–20). Error bars represent bootstrap 95% confidence intervals.

Score spread grew as load grew. Module 1's overall workload had the tightest distribution ($SD = 2.03$) and the narrowest 95% confidence interval ($[6.43, 8.91]$), just 2.48 points from lower to upper bound. Module 2 was nearly twice as dispersed ($SD = 3.70$, CI $[8.63, 13.15]$). The five higher-load modules all had confidence intervals that overlapped each other, clustered between roughly 7 and 13 on the scale. Module 1's CI barely overlapped with those of Modules 2 ($[8.63, 13.15]$), 5 ($[8.26, 12.78]$), and 6 ($[8.65, 12.45]$), with its upper bound (8.91) sitting only 0.26–0.65 points above their lower bounds. Within the subscales, the spread was even wider: Mental Demand SDs went from 4.12 at M1 to 6.82 at M3, which means individual participants were routinely giving ratings ten or more points apart on the same module. Section 5.5 takes up these person-level differences in detail.

Pairwise Wilcoxon signed-rank tests were run for all 15 module pairs with a Bonferroni-adjusted $\alpha = .0033$. None survived. The two largest gaps, M1 versus M5 (about 3.0 points) and M1 versus M6 (about 2.9 points), came closest. But at $n = 9$, even the strongest possible Wilcoxon outcome ($W = 0$, every participant ranked in the same direction) produces an exact p of .004. That is above the corrected threshold. Put plainly, the test could not reject any pair at this sample size no matter what the data looked like.

Dropping P08 from the analysis (see Section 5.1 for the flagged response pattern) changed nothing of substance. The Friedman test for overall workload came back at $\chi^2 = 14.98$, $p = .010$, still significant. Every pattern reported above held.

5.4. Framework Comparison

To compare the two frameworks directly, each participant's scores for the three HOD modules (M1 RGB Basics, M2 Cellular Automata, M4 Fractal Square) were averaged into a single HOD value; the three CDD modules (M3 Convolution, M5 Perceptron, M6 DDPM Basics) were averaged the same way. A paired Wilcoxon signed-rank test then compared these per-participant aggregates at $n = 9$, matching each participant's HOD mean against their own CDD mean. This approach treats each framework as a single observation per participant and separates the between-framework question from the cross-module sequencing effects documented in Section 5.3. It does not control for module position within the day.

HOD overall workload came in at $M = 9.09$ and CDD at $M = 10.43$, a gap of 1.34 points. The test returned $W = 11.0$, $p = .203$, $d = 0.699$. Not significant, but a medium effect. Both facts matter here. A 1.34-point difference, sustained across nine matched participants, is not noise; it falls below what the sample can resolve at this size. Full subscale comparisons and the exit-ticket understanding item appear in Table 5.6; Figure 5.6 plots the overall workload and Q4 distributions side by side.

Table 5.6.: HOD vs. CDD Framework Comparison Across NASA-TLX Subscales and Exit Ticket Understanding ($n = 9$, Wilcoxon Signed-Rank)

Measure	**HOD *M***	**CDD *M***	**Diff**	***W***	***p***	**Sig.**	***d***	**Magnitude**
Overall Workload	9.09	10.43	+1.34	11.0	.203	ns	+0.699	Medium
Mental Demand	9.74	12.74	+3.00	1.0	.016	*	+1.100	Large
Physical Demand	3.33	5.11	+1.78	1.0	.062	ns	+0.778	Medium
Temporal Demand	8.93	10.85	+1.93	9.0	.250	ns	+0.529	Medium
Performance (inv)	16.26	13.81	−2.44	6.0	.055	ns	−0.825	Large
Effort	10.19	11.59	+1.41	8.0	.098	ns	+0.448	Small
Frustration	6.11	8.48	+2.37	7.0	.148	ns	+0.658	Medium
Exit Q4 (understanding)	3.50	3.04	−0.46	6.0	.219	ns	−0.549	Medium

Note. W = Wilcoxon signed-rank statistic (paired by participant, $n = 9$ for TLX measures, $n = 8$ for Exit Q4). HOD = Hands-On Discovery (M1, M2, M4 averaged); CDD = Conceptual Deep-Dive (M3, M5, M6 averaged). Performance is inverted so that higher values indicate greater perceived success. d = Cohen's d; positive values indicate CDD higher. *$p < .05$.

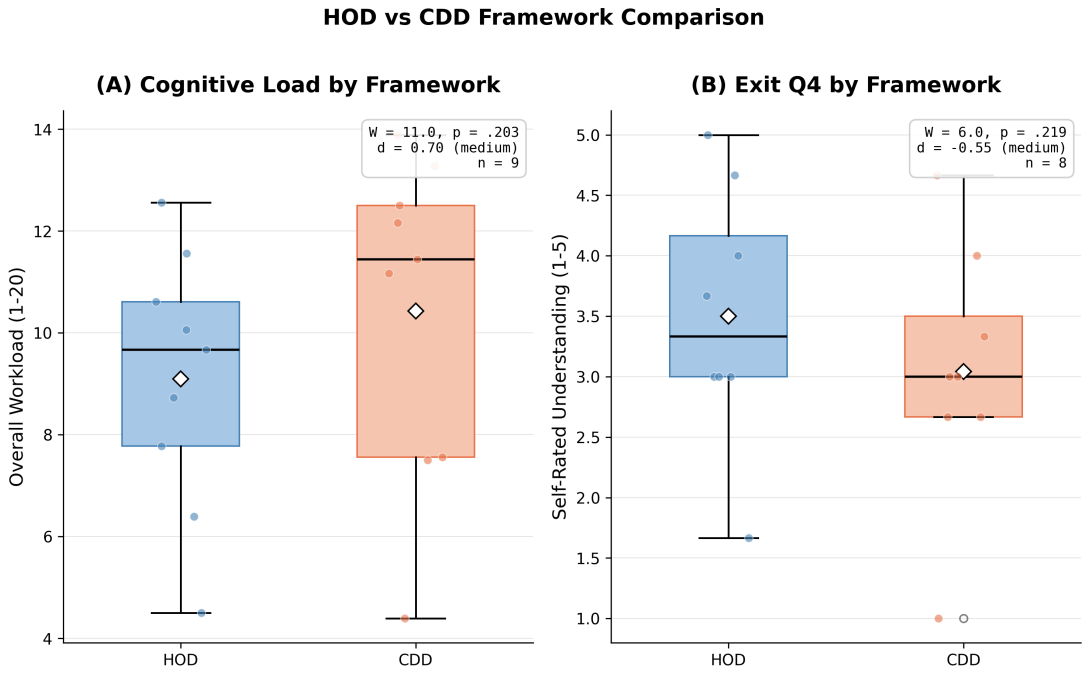


Figure 5.6.: HOD vs. CDD framework comparison ($n = 9$). Panel A: overall workload; Panel B: exit ticket self-rated understanding. Diamonds indicate means; horizontal lines indicate medians.

3.00 points. That is the HOD-versus-CDD gap on Mental Demand: $M = 9.74$ for HOD and $M = 12.74$ for CDD, $W = 1.0$, $p = .016$, $d = 1.100$. Across all seven TLX comparisons, Mental Demand was the only one to cross $\alpha = .05$. The effect is large. Seven of nine participants rated CDD modules higher on Mental Demand than HOD modules; one went the other direction and one (P08) was tied at the scale floor. The Wilcoxon test drops ties, so the effective comparison ran on eight participants with seven favouring CDD, producing the $W = 1.0$ result.

Every other subscale ran numerically higher for CDD, but none reached significance. Physical Demand was the closest runner-up on effect size ($d = 0.778$, $p = .062$). Performance went the other way: HOD participants rated themselves considerably more successful, $M = 16.26$ versus $M = 13.81$, a 2.44-point gap in the direction of HOD, $d = -0.825$, $p = .055$. Large effect, not significant. Frustration was $M = 6.11$ (HOD) against $M = 8.48$ (CDD), $d = 0.658$, $p = .148$. Temporal Demand ($d = 0.529$, $p = .250$) and Effort ($d = 0.448$, $p = .098$) both pointed toward CDD being more demanding without crossing the threshold. Six of the seven comparisons ran in the same direction: CDD higher. One crossed the line.

On the exit-ticket understanding scale (Q4, rated 1–5), HOD averaged $M = 3.50$ and CDD averaged $M = 3.04$, a difference of 0.46 points. That 0.46-point gap produced $W = 6.0$, $p = .219$, $d = -0.549$. Not significant. The negative sign on d records the direction: HOD higher on understanding, matching the workload pattern but on the opposite side of the dependent variable. A medium effect, unresolved by the sample size.

Module-level scores break the CDD aggregate apart. HOD sessions were relatively stable: M1 at $M = 3.67$ ($SD = 1.22$, $n = 9$), M2 at $M = 3.25$ ($SD = 1.28$, $n = 8$), M4 at $M = 3.62$

($SD = 0.92$, $n = 8$), a total range of 0.42 points across three sessions. CDD sessions were not. M3 (Convolution) scored $M = 3.88$ ($SD = 1.46$, $n = 8$), the highest of any module in the workshop, while M5 (Perceptron) dropped to $M = 3.00$ ($SD = 1.41$, $n = 7$) and M6 (DDPM Basics) fell further still, to $M = 2.14$ ($SD = 1.07$, $n = 7$). That is a decline of 1.74 points across three consecutive CDD sessions. M6's mean of 2.14 was the lowest self-reported understanding score in the dataset. Standard deviations were high throughout the CDD sequence ($SD = 1.07$–1.46), indicating substantial disagreement among participants at every stage.

M3 did not fit the CDD pattern. Its exit-ticket score was the highest of any module. That includes all three HOD sessions. Its overall TLX workload ($M = 9.93$) placed it below M2 HOD ($M = 10.94$) and made it the lowest-load session among the three CDD modules. On both understanding and workload, M3 sat closer to the HOD profile than to M5 or M6.

The within-CDD spread on self-rated understanding was 1.74 points, running from M3 at the top to M6 at the bottom. The within-HOD spread was 0.42 points across M1, M2, and M4. Averaging only M5 and M6 gives a CDD understanding mean of $M = 2.57$, which widens the HOD-versus-CDD gap from 0.46 points in the full aggregate to 0.93 points, because the pooled CDD figure of 3.04 is partly a product of M3's atypical score pulling the average up. CDD did not move as a block on self-rated understanding. What separates M3 from M5 and M6 is examined in Section 6.1.

Removing P08 from the framework comparison (see Section 5.1 for the flagging rationale) shifted the overall workload result to $W = 7.0$, $p = .148$, compared with $p = .203$ in the full sample. Still not significant. The direction of every subscale comparison held with P08 excluded, and the non-significant differences changed only slightly in magnitude. The module-level Q4 figures above reflect all available respondents per module ($n = 9$ for M1, $n = 8$ for M2–M4, $n = 7$ for M5–M6). The M3 anomaly on understanding did not depend on P08's responses.

5.5. Individual Differences

The group means in Section 5.3 sit on top of substantial person-to-person differences. Figure 5.7 plots all nine trajectories. P05 stayed above 12 for five of the six modules; P07 exceeded 12 in four. At the other extreme, P08 never broke 6, and P03 remained below 8 in five of six modules, peaking at 9.00 only during the Perceptron session. The spread held across nearly every session: in five of the six modules, at least six points separated the lowest and highest ratings, and twice the gap topped ten. At Module 2, P05 scored 15.17 while P08 scored 5.83. Nearly ten points apart. At Module 5, the spread was just as wide: P08 at 3.67 against P05 at 13.00. Prior technical experience was the obvious candidate, and the correlation analysis backed it up.

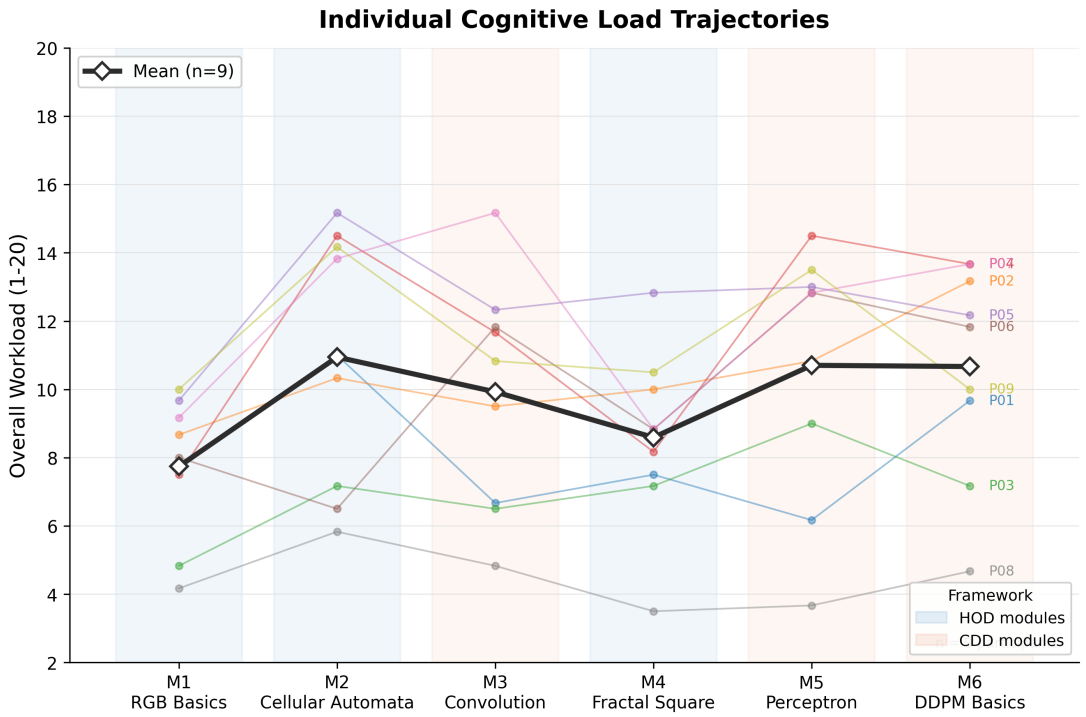


Figure 5.7.: Individual NASA-TLX workload trajectories across six modules ($n = 9$). Thin lines represent individual participants; bold black line shows the group mean. Shading indicates framework (blue = HOD, red = CDD).

$r_s = -.857$, $p = .007$. That was the Spearman coefficient when each participant's composite prior-experience score (see Section 5.1) was set against their mean NASA-TLX workload across six modules ($n = 8$; P04 excluded for missing post-test data). Strongest individual-level finding in the dataset. P03 sat at the top of the experience range (composite = 0.479 on the 0–1 scale) and averaged a TLX of 6.97. P05 and P07, both below 0.10, averaged 12.53 and 12.25. Eight points separated top from bottom. P08 was a partial exception: moderate experience (0.302) paired with the lowest mean TLX in the sample at 4.45, consistent with the floor-rating pattern flagged in Section 5.1. Excluding P08 pushed the coefficient to $r_s = -.893$, $p = .007$. The relationship held. It got tighter.

That pattern did not carry over to learning gains. The Spearman coefficient between prior experience and normalized gain was $r_s = +.204$, $p = .629$, $n = 8$. Flat. A median split put concrete values on the non-result: the lower-experience half (P05, P06, P07, P09; $M_{exp} = 0.112$) averaged $g = 0.305$ (95% CI $[0.21, 0.40]$), while the higher-experience half (P01, P02, P03, P08; $M_{exp} = 0.352$) averaged $g = 0.550$ (95% CI $[0.26, 0.84]$). The raw means differ by 0.245, but the confidence intervals overlap substantially and the rank-order coefficient was nowhere near significance. Experienced participants found the workshop less demanding. They did not learn measurably more.

Section-level Hake gains by experience group added a wrinkle. In Section A (Arrays and Images), the higher-experience group scored $g = 0.771$ against $g = 0.262$ for the lower group, a gap of half a normalized unit. Section B (Computational Concepts) flipped: the lower group reached $g = 0.458$, the higher group only $g = 0.267$. No clear direction. Sections C ($g = 0.617$

vs. 0.417) and D ($g = 0.458$ vs. 0.083) both favored the higher-experience group, but at $n = 4$ per half, no section-level comparison was tested for significance; all four splits are descriptive only.

The full Spearman matrix appears in Table 5.7. At $n = 8$, the weak positive association between cognitive load and normalized gain ($r_s = +.204$, $p = .629$) warrants no interpretation. Figure 5.8 adds a fourth variable to the matrix, average self-rated understanding (Q4 across modules); prior experience and mean Q4 produced $r_s = .755$, $p = .031$, a strong positive link that ran in the same direction as the experience-load finding. Normalized gain was not significantly associated with any of the other three variables.

Table 5.7.: Spearman Rank-Order Correlations Among Individual-Level Variables ($n = 8$)

	Experience	**Avg TLX**	**Norm. Gain**
Prior experience composite	—		
Average NASA-TLX workload	$-.857$**	—	
Normalized gain (Hake g)	$+.204$	$+.204$	—

Note. $n = 8$ (P04 excluded: no post-test). **$p < .01$.

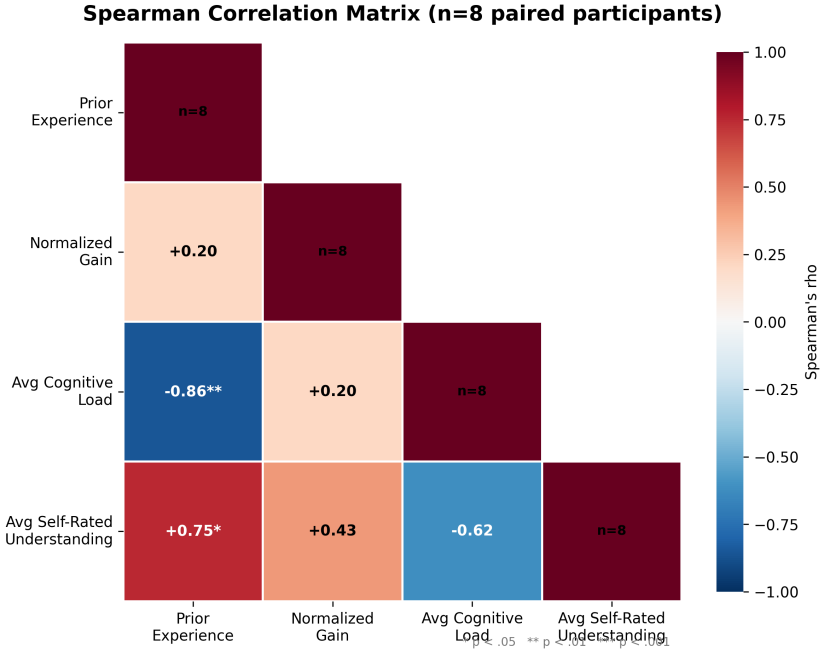


Figure 5.8.: Spearman rank-order correlation matrix ($n = 8$). Variables: prior experience, normalized gain, average cognitive load, and self-rated understanding. *$p < .05$, **$p < .01$.

Pooled across all module-participant pairs where both measures were present ($n = 47$), exit-ticket understanding (Q4) and NASA-TLX workload correlated at $r_s = -.514$, $p < .001$: higher self-rated understanding went with lower workload. The module-level picture was less uniform. M3 (Convolution) sat at one extreme, $r_s = -.932$, $p < .001$ ($n = 8$); at that module the two scores tracked each other nearly one-to-one. M1, M4, and M5 all landed between $r_s = -.600$ and $r_s = -.635$, strong associations but none significant at sample sizes

of seven to nine. M2 ran weaker, $r_s = -.316$ ($n = 8$, $p = .446$). M6 broke the pattern entirely: $r_s = +.037$, $p = .937$ ($n = 7$). Nothing there. How well a participant felt they understood the DDPM material had nothing to do with how hard they found it, a disconnect seen in no other module. Both M3 and M6 were CDD sessions, yet one had the tightest Q4-TLX coupling in the dataset and the other had none. Table 5.8 gives the full module-level breakdown.

Table 5.8.: Module-Level Spearman Correlations Between Self-Rated Understanding (Q4) and NASA-TLX Workload

Module	**Framework**	***n***	r_s	***p***	**Sig.**
Pooled (all modules)	—	47	$-.514$	$< .001$	***
M1 RGB Basics	HOD	9	$-.635$	.066	ns
M2 Cellular Automata	HOD	8	$-.316$	.446	ns
M3 Convolution	CDD	8	$-.932$	$< .001$	***
M4 Fractal Square	HOD	8	$-.623$	.099	ns
M5 Perceptron	CDD	7	$-.600$	.154	ns
M6 DDPM Basics	CDD	7	$+.037$	.937	ns

Note. Q4 rated on a 1–5 scale (higher = better understanding). NASA-TLX on a 1–20 scale (higher = greater workload). Pooled r_s computed across all module-participant pairs where both measures were present. ***$p < .001$.

5.6. Qualitative Findings

Exit ticket responses to questions Q1–Q3 and Q5 were analysed using reflexive thematic analysis (see Section 3.7). Across 47 exit tickets ($n = 9$ participants, six modules), 158 responses received codes drawn from a 30-code inductive codebook organised into five categories: Learning, Connections, Challenges, Engagement, and Experience, for a total of 204 code applications (multiple codes were permissible per response). Table 5.9 summarises the three most frequent codes per category; Figure 5.9 plots the full code frequency distribution.

Table 5.9.: Qualitative codebook: top three codes by frequency and total applications per category.

Category	**Top three codes (applications)**	**Total**
Learning (5 codes)	LEARN_CONCEPTUAL (31); LEARN_PROCEDURAL (10); LEARN_TOOL_SETUP (5)	51
Connections (6 codes)	CONN_EXPLICIT (16); CONN_NONE (8); CONN_VAGUE (6)	38
Challenges (10 codes)[a]	CHAL_CONCEPTUAL (12); CHAL_TECH_SETUP (8); CHAL_NONE (7)	52
Engagement (6 codes)[b]	ENG_DEPTH (14); ENG_CONTINUE (10); ENG_BREADTH (6)	41
Experience (3 codes)	EXP_BEGINNER (10); EXP_HANDS_ON (7); EXP_NOVELTY_POS (5)	22
Total		**204**

Note. Frequencies derive from `theme_frequencies.csv` (Phase 5 analysis output, 2026-02-26). Multiple codes were permissible per response; the total (204) therefore exceeds the number of coded responses (158). Category totals include all codes in each category, not only the three listed. [a] Includes one residual code (CHAL_PROCEDURAL, $n = 1$) not assigned a category in the coding log. [b] Includes one residual code (ENG_SATISFIED, $n = 1$) not assigned a category in the coding log.

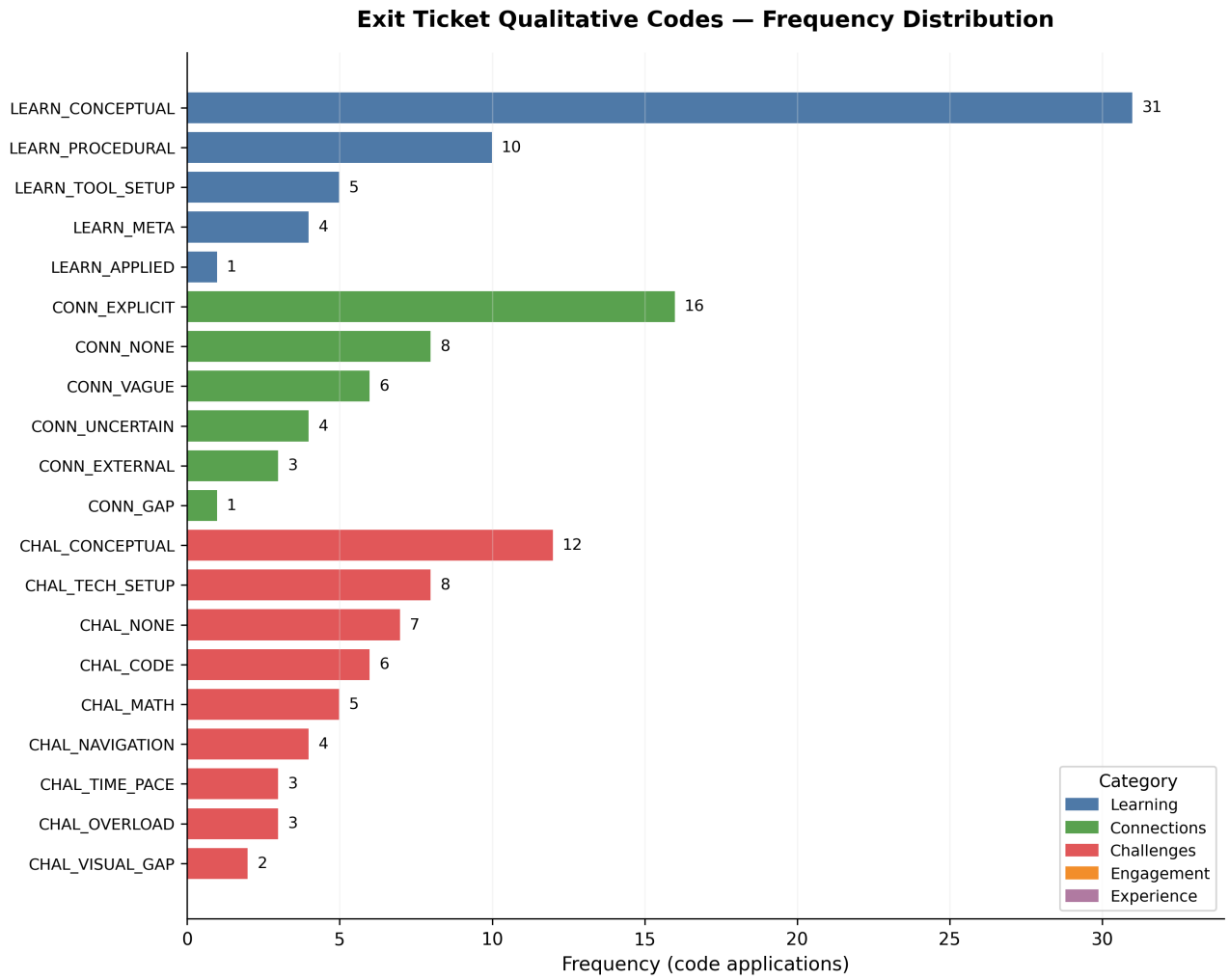


Figure 5.9.: Frequency distribution of qualitative codes from exit ticket responses. Bars are grouped by category: Learning (blue), Connections (green), Challenges (red), and Engagement (orange).

LEARN_CONCEPTUAL led the codebook with 31 applications and was one of only two codes applied in every module (the other being ENG_DEPTH, with 14). Responses named specific technical content across the full curriculum range, from array operations in M1 to perceptron logic in M5: “Array slicing and the difference between height/width, when the width and height are reversed in order. Also channels and greyscale” (P09, M1); “A perceptron decides binary values depending on input and step (activation function)” (P02, M5). In M5 and M6, LEARN_META codes also emerged (4 applications, Conceptual Deep-Dive (CDD) modules only), where participants reflected on knowledge gaps rather than naming content: “I didn’t yet know a lot about neural networks” (P03, M5). LEARN_TOOL_SETUP accumulated 5 applications across the workshop, spiking at M1 and M6. The logic is different in each case: where the first module introduced the Python environment, the last put PyTorch installation ahead of diffusion model concepts. “My CPU is slow” (P03, M6) was one participant’s entire M6 Q1 response.

Only two of the seven participants who completed M6 exit tickets recorded explicit connections, and CONN_NONE codes (8 applications) spread across M2–M6, with two participants

(P07 and P08) accounting for most instances through persistent difficulty making connections. The curriculum gap compounded this pattern in later modules: the workshop skipped modules 5–8 of the sequence before advancing to the Perceptron unit, and Q2 responses registered the disruption directly. “We jumped a lot of modules, no direct connection” (P02, M6); a second participant reported “couldn’t connect to any concepts yet” (P07, M5). Through M2–M4, by contrast, CONN_EXPLICIT dominated (12 of its 16 total applications fell in those three modules); arrays were the most-referenced prior concept, cited by four participants, while kernels bridged M2 to M3 for two further participants (Figure 5.10).

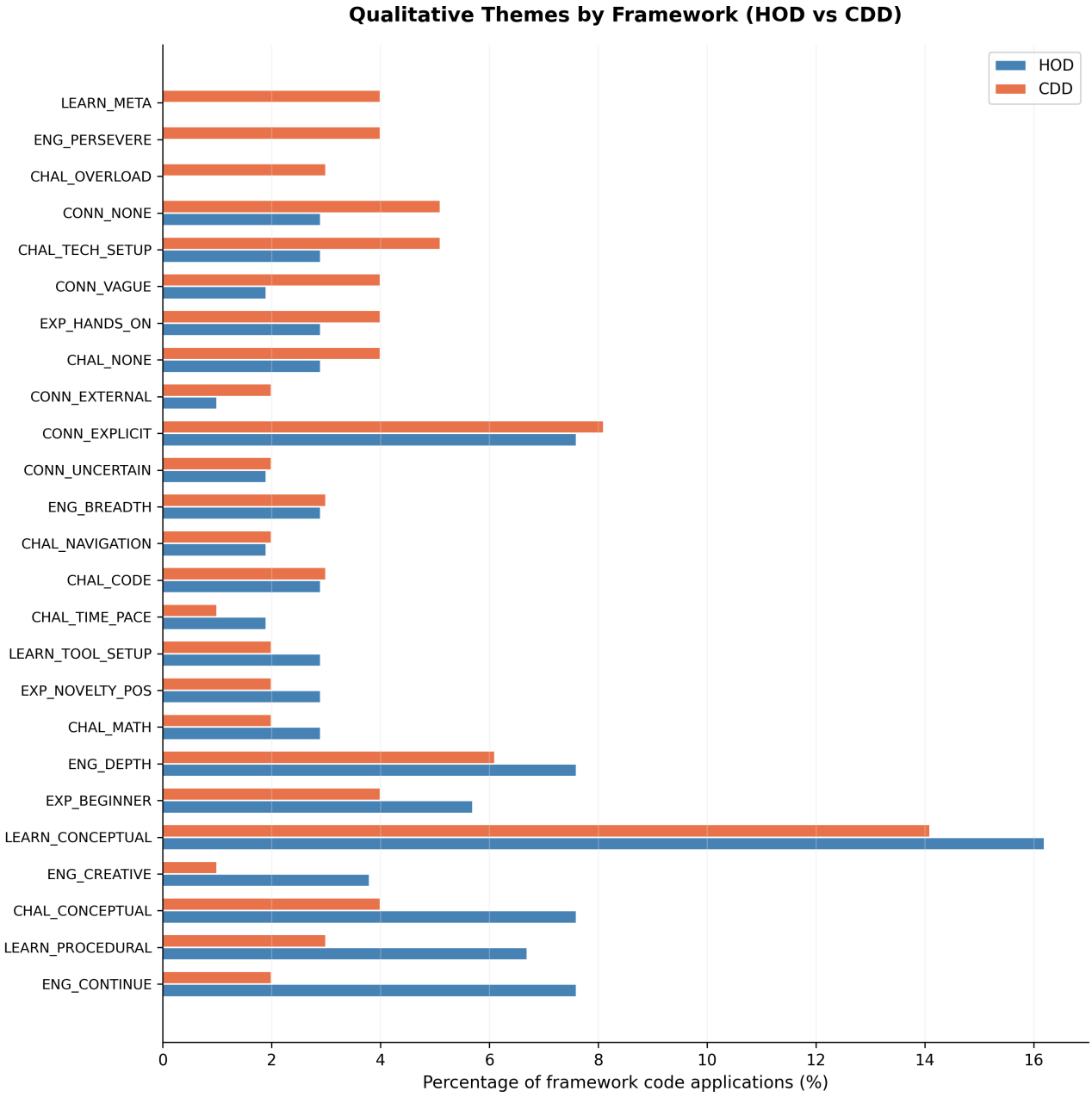


Figure 5.10.: Distribution of qualitative codes by pedagogical framework. Blue bars represent HOD modules; red bars represent CDD modules.

CHAL_CONCEPTUAL was the most frequent challenge code overall (12 applications) and appeared across both frameworks (8 in Hands-On Discovery (HOD), 4 in Conceptual

Deep-Dive (CDD)). The two frameworks differed in secondary challenge types: Hands-On Discovery (HOD) modules drew execution-level complaints: missing visualisation steps (CHAL_VISUAL_GAP, 2 applications, M2 only) and navigation problems (CHAL_NAVIGATION). Conceptual Deep-Dive (CDD) modules, by contrast, uniquely generated overload codes (CHAL_OVERLOAD, 3 applications, all in M5) and the sole copy-paste criticism. CHAL_TECH_SETUP (8 applications) appeared in both frameworks but spiked in M6 (4 applications) owing to PyTorch installation failures rather than pedagogical design. "The math functions are very unfamiliar to me (computational math is confusing)" (P05, M5) and "Too much reading, complex content" (P09, M5; see also Section 5.7) were representative Q3 responses in M5. One response addressed a different concern: "The exercises are copy-paste without understanding" (P09, M5), the only application of CHAL_COPY_PASTE in the dataset.

At $M = 10.94$, Module 2 (Cellular Automata) carried the highest NASA-TLX workload among Hands-On Discovery (HOD) modules, an anomaly for a visual-first unit. Q3 responses identified two concurrent causes. Two participants independently noted the absence of intermediate visualisation: "Maybe in the first exercise, you could add a piece of code that could visualise each step of the transition because I feel like we expected it" (P01, M2) and "There was no display of the intermediate grid states in the output of the exercise" (P02, M2). Content density was a separate factor: "information is very condensed in this module" (P05, M2). No other Hands-On Discovery (HOD) module drew this combination of structural complaints.

The highest exit ticket self-rated understanding across the workshop came from Module 3 (Convolution), at $M = 3.88$, and qualitative responses matched that picture. Participants with prior exposure made explicit connections in M3: "Padding is a smart idea. I had this topic in a class, we just skipped the edges" (P03, M3). P07 was different here. This participant, who rated understanding at 1 or 2 in every other module, found an applied connection in M3: "Kernel values gave me a lot of insights how creative apps could be working behind the scenes" (P07, M3).

Across M1–M4, Q5 exploration interests ran toward creative extension: "fractal patterns, obviously" (P03, M4) and "Mandelbrot, depth and recursion call" (P05, M4) were typical responses. By M5 and M6, ENG_CREATIVE disappeared after M4 and ENG_CONTINUE dropped from its early peaks (4 in M1, 3 in M2) to zero in M5. In their place, ENG_PERSEVERE emerged exclusively in Conceptual Deep-Dive (CDD) modules (M3 = 1, M5 = 1, M6 = 2); ENG_DEPTH persisted at its baseline level (M5 = 2, M6 = 2) but no longer dominated. "I just want to keep exposing myself to these concepts until I get there" (P07, M5) and "Hoping I will recall these experiences in future, once I dive deeper into these topics" (P07, M6) capture the late-module tone. One M6 response stood apart: "Going through all the chapters in order" (P02, M6), the sole application of ENG_SEQUENTIAL in the dataset.

5.7. Mixed Methods Integration

The quantitative and qualitative data strands ran concurrently during the workshop and were integrated post-hoc through a convergence matrix. Strand A (NASA-TLX workload ratings and pre/post knowledge scores) and Strand B (exit ticket thematic analysis, $n = 158$ coded responses, 30-code codebook) formed the primary instruments. A post-workshop evaluation form completed by seven of the nine participants provides supplementary descriptive data; it was not part of the original instrument set and its responses are treated as corroborating rather than primary evidence.

Table 5.10 maps five convergence points across the three strands.

Table 5.10.: Mixed methods integration matrix: convergence of quantitative, qualitative, and supplementary evidence across five themes.

Theme	Strand A: Quantitative	Strand B: Qualitative	Strand C: Feedback form	Assessment
Learning gains	$W = 0.0$, $p = .008$, $d = 1.615$; all 8 paired participants improved	LEARN_CONCEPTUAL: 31 applications (most frequent code, all modules)	C1 learned new concepts: $M = 5.00$, $SD = 0.00$ (ceiling, $n = 6$)	Convergent
Cognitive overload at pace	TLX trajectory: $\chi^2(5) = 15.795$, $p = .008$, $W = 0.351$; CDD mean $M = 10.43$	CHAL_OVERLOAD: 3 applications (all M5); total challenge codes in M5 = 10	Pace: 5 of 7 rated "slightly fast" or "too fast"	Convergent
Transfer gap	Section D gain $g = 0.271$ (lowest); $p = .062$ (ns)	CONN_NONE: 8 applications across M2–M6; curriculum gap reinforced perceived disconnect at M5–M6 transition	C3 confident applying to own projects: $M = 3.50$, $SD = 1.22$ ($n = 6$)	Convergent
Visual-first engagement	HOD workload $M = 9.09$ vs. CDD $M = 10.43$; HOD exit understanding $M = 3.50$ vs. CDD $M = 3.04$	ENG_CREATIVE codes concentrated in HOD modules (4 of 5); ENG_DEPTH most frequent in M4	C2 visual examples helped: $M = 4.83$, $SD = 0.41$ ($n = 6$)	Convergent
Module coherence gap (M5–M6)	M6 exit understanding $M = 2.14$ (lowest); Section D gain trend not significant	CONN_GAP and CONN_NONE in M5–M6; P02: "We jumped a lot of modules, no direct connection"	C4 progression made sense: $M = 3.83$, $SD = 1.17$ ($n = 6$)	Partial

Note. Strand A = NASA-TLX workload ratings and pre/post knowledge test ($n = 9$ for TLX, $n = 8$ paired for knowledge test). Strand B = exit ticket thematic analysis (30-code codebook, 158 coded responses). Strand C = post-workshop evaluation form (supplementary, $n = 7$). HOD = Hands-On Discovery (HOD); CDD = Conceptual Deep-Dive (CDD). ns = not significant.

The heatmap in Figure 5.11 maps qualitative code frequency across modules and provides visual support for the convergence patterns summarised in Table 5.10.

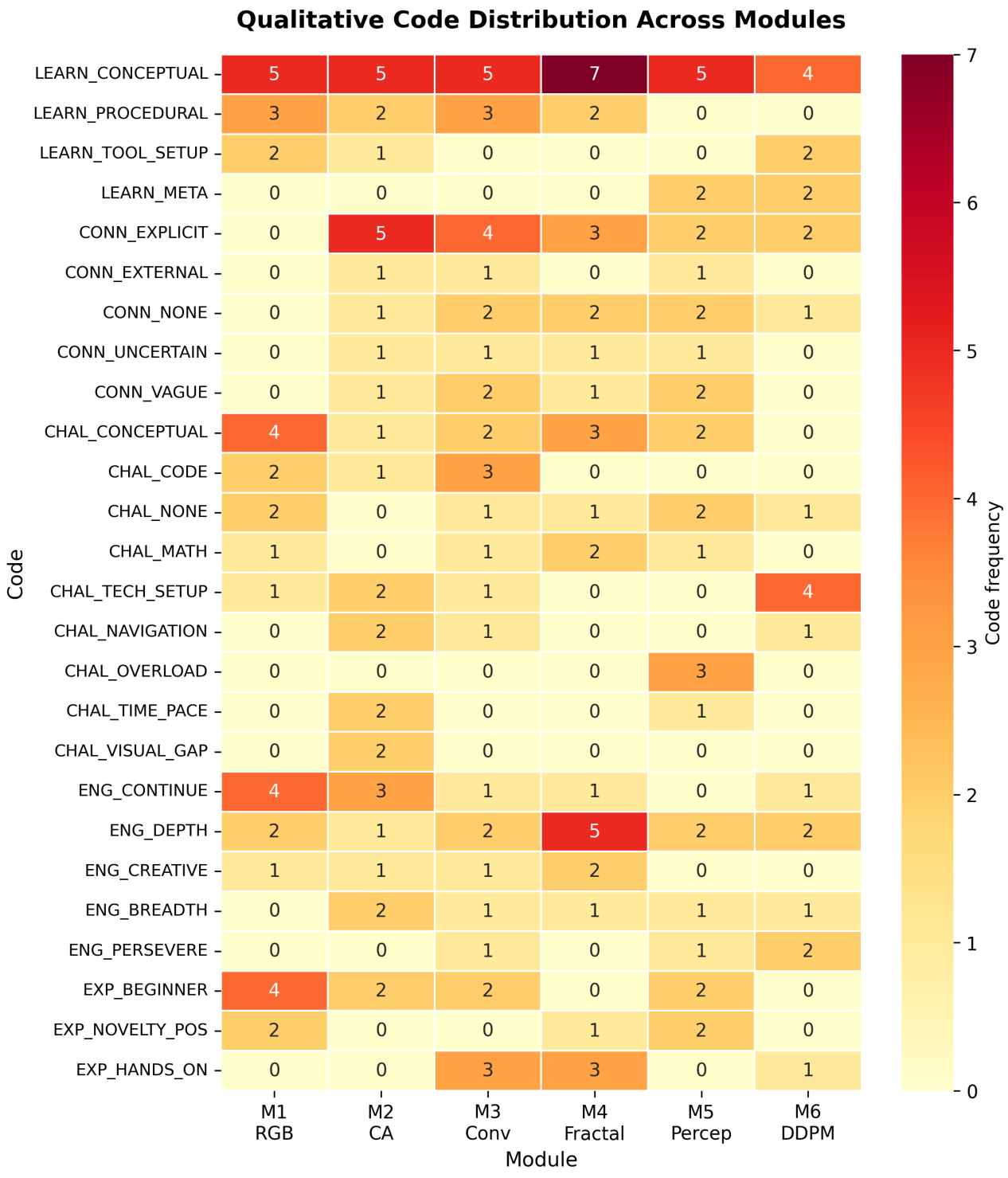


Figure 5.11.: Heatmap of qualitative code frequency across six workshop modules. Colour intensity indicates frequency of code application (white = 0, dark red = 7+).

CHAL_OVERLOAD codes appeared exclusively in M5 and ENG_CREATIVE codes were absent from M5 and M6, while CONN_NONE spread across M2–M6 but co-occurred with the curriculum gap at M5–M6, reinforcing the strand B evidence across the convergence points.

Across all three strands, learning gains produced the clearest convergence. Knowledge scores improved significantly from pre- to post-test ($W = 0.0$, $p = .008$, $d = 1.615$), with all eight paired participants recording higher post-test scores. Exit ticket coding matched this picture: LEARN_CONCEPTUAL was the most frequently applied code in the entire codebook (31 applications), distributed across all six modules, from basic array operations in M1 through perceptron logic in M5. Feedback form item C1 recorded a ceiling response ($M = 5.00$, $SD = 0.00$, $n = 6$), with every respondent giving the maximum possible rating.

Cognitive load distribution showed equally clear agreement across strands. The NASA-TLX trajectory varied significantly by module ($\chi^2(5) = 15.795$, $p = .008$, $W = 0.351$), with Conceptual Deep-Dive (CDD) modules carrying higher mean workload. Exit ticket coding produced three applications of CHAL_OVERLOAD, all in M5; one participant wrote: “Too much reading, complex content. Unfortunately, I feel like I didn’t learn much because there is too much to learn” (P09, M5). Five of seven feedback form respondents rated the session pace as “slightly fast” or “too fast.” Workload was manageable through M4 but accumulated to problematic levels as the curriculum transitioned from Hands-On Discovery (HOD) to Conceptual Deep-Dive (CDD) modules.

Transfer to generative AI contexts was the weakest link in the learning outcomes, and all three strands registered this consistently. Section D (Generative AI) recorded the lowest normalized gain ($g = 0.271$) and the gain did not reach significance ($p = .062$). CONN_NONE codes (8 applications total) appeared across M2–M6, with the curriculum gap reinforcing connection difficulties in M5 and M6 where participants reported an inability to relate new content to earlier modules. Feedback form item C3 (confident applying concepts to own projects) scored $M = 3.50$ ($SD = 1.22$, $n = 6$), tied for the lowest of the Section C learning outcome items (alongside C5) and well below C1 ($M = 5.00$) and C2 ($M = 4.83$). Knowledge acquisition and transfer confidence thus tracked in opposing directions: transfer readiness was lowest where content difficulty peaked.

One partial convergence warrants note. Qualitative codes and feedback form data both signal a coherence gap at the M5–M6 transition: CONN_GAP appeared exclusively in M6, and the feedback form’s C4 mean ($M = 3.83$, $SD = 1.17$) sat below the Section C ceiling items (C1, C2) though above C3 and C5. The quantitative record is less definitive. M6 exit understanding fell to its session low ($M = 2.14$) and Section D gains remained below the medium-gain threshold, but neither result reached statistical significance. This constitutes a partial rather than full convergence: the qualitative and supplementary strands are consistent in direction, while the quantitative signal, though pointing the same way, did not reach significance.

Divergences between strands and the limitations they introduce are taken up in Section 6.7.

5.8. Instrument Quality

Two instruments generated data amenable to psychometric evaluation: the 24-item multiple-choice knowledge test, administered before and after the workshop, and the NASA Task Load Index, completed after each of the six modules.

5.8.1. Knowledge Assessment Reliability

Internal consistency for the pre-test, calculated on the full entry sample ($n = 9$, $k = 24$), returned KR-20 $= 0.742$, above the 0.70 acceptability floor. Post-test consistency rose to KR-20 $= 0.913$ on the paired subsample of eight, a value classified as excellent (Table 5.11). The gap between the two estimates is partly mechanical. Pre-test scores clustered near the scale floor: across all nine participants the group mean was 3.22 out of 24, the median was 2, and 74% of item responses carried an "I don't know" (IDK) selection. Floor effects compress inter-individual variance and dampen item-level covariance, both of which suppress KR-20 estimates regardless of true instrument quality. On the post-test ($n = 8$), the IDK rate fell to 24%, the standard deviation of the paired subsample widened from 2.72 to 6.29, and greater score spread allowed item discrimination to operate more fully. Section D retained the highest residual IDK rate; the subsection figures appear in Table 5.11.

Table 5.11.: Knowledge Assessment Internal Consistency (KR-20) by Administration and Scope ($n = 8$–9, $k = 6$–24)

Administration	Scope	*n*	*k*	KR-20	Benchmark
Pre-test	Full test (Sections A–D)	9	24	0.742	Acceptable
Post-test	Full test (Sections A–D)	8	24	0.913	Excellent
Post-test (sub)	Section D only (Q19–Q24)	8	6	0.919	Excellent

Note. k = number of items. Benchmark thresholds: ≥ 0.70 acceptable, ≥ 0.90 excellent (Nunnally 1978). $n = 9$ for pre-test (full entry sample); $n = 8$ for post-test (P04 excluded: no post-test data). The Section D subsection row is reported separately because that section retained a 56% IDK rate at post-test; the subsection KR-20 reflects only participants who responded to those six items.

5.8.2. Item Difficulty and Discrimination

Mean item facility rose from 0.134 on the pre-test to 0.505 on the post-test, a shift of $+0.371$ across all 24 items (Figure 5.12). Twenty-two of the 24 items became easier to answer correctly after instruction. Two items diverged from this pattern. Q3, drawn from Section A (Arrays and Images), returned a facility decrease of 0.292, the largest negative shift in the item set. Q17, from Section C (Neural Networks), fell by 0.097. Both items appeared on parallel but non-identical pre- and post-test forms; the difficulty changes reflect differences in item phrasing and content coverage between versions rather than post-instruction regression

on a common item. The Q3 and Q17 answer keys were independently verified during data validation and found to be correct.

Five items retained a post-test IDK rate of 50% or above; all five belonged to Sections C and D. This pattern aligns with the normalized gain data in Section 5.2: the curriculum's final two content blocks produced the smallest score increases and the highest residual uncertainty. At post-test, 17 of the 24 items produced point-biserial discrimination indices of $r_{it} \geq 0.20$. The remaining seven fell below that threshold: three were Section A items whose near-ceiling post-test facility (≥ 0.875) compressed variance, and four were Section B items where moderate facility did not translate into discriminating power. Section D items, despite high IDK rates, all exceeded the $r_{it} \geq 0.20$ threshold because the participants who did respond showed consistent score patterns.

5.8.3. NASA-TLX Data Completeness

Across nine participants and six modules, the NASA-TLX produced 54 complete rating sets. The instrument covers six subscales (Mental Demand, Physical Demand, Temporal Demand, Performance, Effort, and Frustration), each scored on a 1–20 range; overall workload was the unweighted mean of all six. Nothing was missing. Every participant completed every module rating, for a subscale-level completeness rate of 100%, against an overall study figure of 94.1% (127 of 135 data cells; see Section 5.1). Both shortfalls in the broader dataset sat outside the TLX: one participant left before the post-test, and a second had no exit ticket data for the final two modules. No imputation was applied to any dataset.

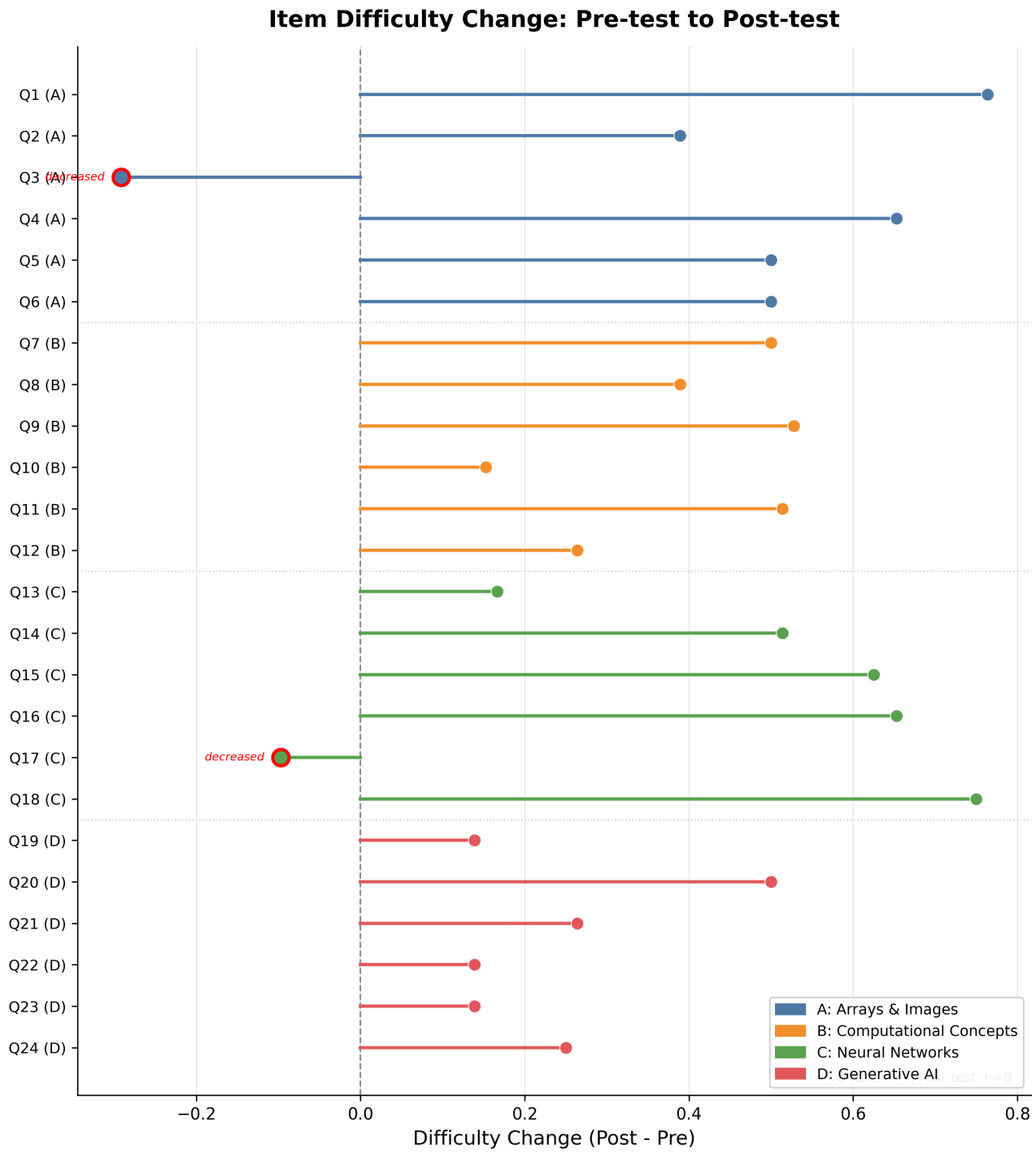


Figure 5.12.: Change in item facility from pre-test to post-test for all 24 knowledge assessment items, grouped by content section. Positive values indicate improved performance; red circles mark items with decreased facility.

6. Discussion

The workshop produced three converging findings. Participants gained substantially in pre-to-post knowledge, with the overall effect size placing well within the large range. Cognitive load varied significantly across the six modules; mental demand, rather than time pressure or physical effort, drove most of that variation. Prior programming experience correlated strongly and negatively with perceived workload, an association strong enough to hold across a sample of nine despite the conservative power available.

6.1. RQ1: Framework Design Principles

The pre-to-post knowledge gain is the clearest single piece of evidence bearing on RQ1. Every participant improved. A Wilcoxon signed-rank test of the full pre-to-post contrast returned $W = 0.0$, $p = .008$, $d = 1.615$, a large effect, and the gains held across all four content sections (see Section 5.2). Attributing that outcome to one framework alone, however, would exceed what the data can support. Hands-On Discovery (HOD) and Conceptual Deep-Dive (CDD) modules were interleaved across the day, and the knowledge instrument measured cumulative learning rather than per-module acquisition. The gain reflects the sequence as a whole. The more precise question the framework comparison can answer is how the two approaches differ in the cognitive experience they create, and what that difference implies for design.

The comparison ran in the same direction on seven of eight measured dimensions, all favouring Hands-On Discovery (HOD) on accessibility. On overall workload, Hands-On Discovery (HOD) averaged $M = 9.09$ against Conceptual Deep-Dive (CDD)'s $M = 10.43$, a 1.34-point gap carrying a medium effect ($d = 0.699$, $p = .203$) that the sample size could not resolve. The one comparison to cross the significance threshold was mental demand: Hands-On Discovery (HOD) $M = 9.74$ versus Conceptual Deep-Dive (CDD) $M = 12.74$, $W = 1.0$, $p = .016$, $d = 1.100$. Seven of nine participants ranked Conceptual Deep-Dive (CDD) higher on this subscale (one tied at the floor). Exit-ticket self-rated understanding (Q4) followed the same direction: Hands-On Discovery (HOD) $M = 3.50$, Conceptual Deep-Dive (CDD) $M = 3.04$, $d = -0.549$, $p = .219$. Performance and Frustration subscales likewise favoured Hands-On Discovery (HOD) at medium-to-large effect sizes without reaching significance (see Table 5.6). Cognitive load theory holds that extraneous processing competes with schema acquisition for finite working memory capacity (Sweller 1988); instructional formats that ground new material in observable output before introducing abstract notation can shift

the balance toward encoding rather than decoding. The Hands-On Discovery (HOD) structure, built around visible array transformations that precede formal operator names, is designed around exactly that principle. (Mayer 2014; Papert 1980)

M3 did not fit the Conceptual Deep-Dive (CDD) pattern. A Conceptual Deep-Dive session on kernel convolution, it returned the highest exit-ticket understanding score of any module in the workshop, $M = 3.88$, while its overall workload ($M = 9.93$) sat below M2 Hands-On Discovery (HOD) ($M = 10.94$). The design of M3 explains the anomaly. The session began with participants running a kernel window across an image in code and observing the transformed output before the convolution operator was formally named. The abstraction arrived second, grounded in a visual referent the code had already produced. That sequencing inverts the typical Conceptual Deep-Dive (CDD) pattern, where notation generally precedes application. Vygotsky (1978) describes this form of bridging as instruction operating within the zone of proximal development, where incoming material connects to something the learner can already partially manage; (Wood et al. 1976) M3's visual entry point served that role. M5 and M6 did not replicate it, and their understanding scores sit 0.74 to 1.74 points below M3 on the same measure.

Section D of the knowledge test, covering Generative AI, produced the weakest gains in the battery: $g = 0.271$, $p = .062$, below the significance threshold. Residual "I Don't Know" responses for Section D items stood at 56% post-test. Most participants lacked sufficient footing to attempt an answer. Exit ticket data corroborate the picture: CONN_NONE codes (8 applications), marking responses where participants reported no conceptual link between a session's content and earlier modules, appeared across M2–M6 but were compounded in the final two sessions by the curriculum gap itself. A structural explanation sits in the workshop design itself. The full 15-module curriculum places Modules 5 through 8, covering intermediate neural network architectures, between the procedural foundations of Modules 1 to 4 and the generative model content of Modules 9 to 12. (Bruner 1960) The single-day workshop omitted those four modules. Their absence created a gap in the scaffolding chain that participants registered as a sudden difficulty spike; the module-level workload data in Section 5.3 show the corresponding rise at M5. This is a constraint of the workshop format. The full curriculum was designed to fill those steps across a longer instructional arc.

The framework comparison, the M3 anomaly, and the Section D gap point toward two design principles. First, framework selection should track content type and visual affordance rather than module position alone: Hands-On Discovery (HOD) suits procedural and visually grounded topics, where immediate output substitutes for formal notation; Conceptual Deep-Dive (CDD) suits theory-heavy content, but only when a perceptual or operational anchor precedes the abstraction. M3's $M = 3.88$ understanding score against M6's $M = 2.14$ captures the consequence of that distinction within a single framework. Second, sequential curriculum integrity is a prerequisite for the scaffolding logic to operate across sessions. When incoming material has no prior knowledge to attach to, each new concept must be constructed from nothing; (van de Pol et al. 2010) the workshop's non-adjacent module sequence placed

participants in exactly that position for the generative AI content. Practical recommendations for adapting both principles to time-constrained delivery appear in Section 6.8.

6.2. RQ2: Cognitive Load Management

At $r_s = -.857$, $p = .007$, the correlation between prior programming experience and average NASA-TLX workload was the strongest individual-level result the study produced. With $n = 8$, a Spearman coefficient of that size leaves little room for alternative explanation; the rank ordering of participants by experience almost perfectly inverted their rank ordering by perceived load. Schema theory accounts for the mechanism. (Sweller 1988) Learners who arrive with compiled mental structures for programming constructs do not need to hold each operator, variable name, and control-flow step in working memory at once. A novice and an experienced coder sit through the same session, but the experienced coder chunks an array operation into one retrieval act and moves on. What the NASA-TLX registered was the gap between those two processing states. (Hart et al. 1988)

Workload was not constant across the six modules. A Friedman test on overall TLX scores returned $\chi^2(5) = 15.795$, $p = .008$, $W = 0.351$, a moderate effect showing that decomposition choices shaped the load participants experienced. The trajectory was not monotonic. M1 sat at the floor ($M = 7.74$), M2 spiked to $M = 10.94$, M3 dropped to $M = 9.93$, and M4 continued down to $M = 8.59$; then M5 and M6 closed the day at $M = 10.70$ and $M = 10.67$. Two features stand out. M4, an Hands-On Discovery (HOD) session on fractal recursion, came in 2.35 points below M2 (also Hands-On Discovery (HOD), covering cellular automata) despite both sessions requiring participants to write procedural code and watch emergent visual output. Three hours of accumulated exposure to the workshop's coding rhythm appear to have lowered the Hands-On Discovery (HOD) baseline. The Conceptual Deep-Dive (CDD) modules showed no such recovery. M5 and M6 both sat above 10.5, matching or exceeding the M2 spike. At M6, however, setup friction and accumulated fatigue blurred the line between content-driven load and technical obstacles (Section 3.4.3). Mental demand, rather than time pressure or physical effort, drove the variation: $\chi^2(5) = 24.827$, $p < .001$, $W = 0.552$, a large effect and the strongest concordance of any subscale (see Table 5.5). Where sessions asked participants to assemble new conceptual structures instead of applying familiar procedural patterns, load rose regardless of position in the day.

Set that correlation next to a second one and the picture sharpens. Prior experience did not predict learning gains: $r_s = +.204$, $p = .629$, non-significant. Novices and experienced participants alike improved on the knowledge test. They just worked under different cognitive conditions. Sweller et al. (2019) characterise this split as the expertise reversal effect: scaffolding pitched at low prior knowledge turns into redundant processing for advanced learners, while removing it leaves novices without support. The present data fit that pattern. The curriculum ran a single track; experienced participants found it manageable, novices found it taxing, and both groups learned.

Feedback form responses ($n = 7$, supplementary) back up the numbers. Five of seven respondents rated the workshop pace as Slightly Fast or Too Fast on Q6. P03 wrote, "The AI stuff was a bit fast-paced for someone w/o prior knowledge." P05 offered a parallel observation: "Could break down the modules more for beginners." Neither respondent sat in the high-experience portion of the sample. Their TLX scores placed in the upper half of the distribution, and their complaints landed on exactly the modules (M5, M6) where the trajectory data peaked. The load experience was not spread evenly. Participants felt it and said so.

One design principle follows from the convergence of correlation, trajectory, and participant voice. Managing cognitive load across a progressive curriculum calls for two mechanisms working together: content decomposition, addressed by the guided practice progression and the Hands-On Discovery (HOD)/Conceptual Deep-Dive (CDD) framework selection already discussed in Section 6.1, and learner-level differentiation, which the current single-track design does not supply. (Sweller et al. 2019) That $r_s = -.857$ is not a flaw. It is a structural feature of any mixed-ability group working through uniform material. Addressing it means building adaptive pathways, optional scaffolding layers, or experience-based grouping into the next iteration of the curriculum. Section 6.8 takes up specific recommendations.

6.3. RQ3: TouchDesigner Integration

The workshop tested six modules, all built around Python and NumPy in a browser-based coding environment. None involved TouchDesigner. RQ3, which asks how real-time visual systems can be woven into a progressive AI curriculum, therefore cannot be answered with the data collected on February 12; the design rationale embedded in the full 15-module curriculum and its theoretical coherence against the workshop findings are what this section can assess.

Three curriculum nodes carry the TouchDesigner content. Module 10 introduces the node-based environment and its signal-flow logic, Module 11 pairs that environment with Python scripting inside TD's Script CHOP and Script SOP operators, and Module 13 builds a hybrid pipeline in which a neural style-transfer model trained in earlier modules feeds real-time frames into a TD render graph. The placement tracks the guided practice progression: (Bruner 1960) Modules 1 through 4 build fluency with array manipulation and image processing in pure Python, Modules 5 through 8 move to neural network construction from first principles, and only then does a node-based visual environment appear. The boundary is intentional. Learners' Python scripts run inside a real-time signal graph at that point, not before. The sequencing is deliberate. A TOP node applying a convolution filter parallels the kernel operation coded in Module 3; a CHOP chain normalising sensor input parallels the array reshaping practised in Module 1. What learners face is a shift from text-based to visual-graph representation of operations they already know, not an encounter with wholly unfamiliar material. Perkins et al. (1992) call this low-road transfer: a learner who has practised a procedure across enough varied settings begins to recognise the underlying structure without

deliberate effort. Eight modules spanning four content domains are meant to accumulate that practice base before the representational shift occurs.

No TouchDesigner session ran, but two workshop observations bear on the integration logic indirectly. In the post-workshop feedback forms ($n = 7$), P01 and P03 both named TouchDesigner, unprompted, as the topic they most wanted to see next (Q12). The Python-first modules, it appears, did not merely teach isolated skills; they generated visible demand for the visual-system layer the later curriculum was built to supply. The second piece of indirect evidence comes from M3. That session's anomalous success, discussed in Section 6.1, rested on a visual anchor preceding the formal abstraction. TD integration applies the same principle at curriculum scale: code-based understanding first, then a visual environment that re-represents those operations in real time. (Papert 1980) The parallel is structural.

Empirical validation of the TD pathway sits in a future DBR iteration covering Modules 9 through 13 with instrumentation comparable to the present study. For RQ3, the contribution this cycle can claim is the rationale, not the proof. Section 6.7 catalogues the scope constraints that kept TouchDesigner out of the workshop, and Section 6.8 sets out planned next steps for the second iteration.

6.4. RQ4: Assessment and Evaluation

The 24-item knowledge test returned a post-test internal consistency of KR-20 $= 0.913$, a value that sits above the 0.90 applied-decision threshold (Nunnally 1978) and represents a substantial improvement over the pre-test figure of 0.742. Item analysis reinforced the picture: 17 of the 24 post-test items produced point-biserial discrimination indices at or above the $r_{it} \geq 0.20$ floor (Ebel et al. 1991), and average item facility shifted by $+0.371$ from pre- to post-administration. For a first-iteration instrument developed within a single DBR cycle, those figures are unusually clean. The seven items that fell below the discrimination threshold sat in Sections A and B, where near-ceiling facility or balanced difficulty compressed the variance needed for the statistic to operate. Section D items, despite carrying the highest residual IDK rates, all exceeded the threshold because participants who did respond showed consistent score patterns. Instrument quality thus reflected response variance rather than content exposure alone.

Exit-ticket self-rated understanding (Q4) and NASA-TLX workload offer a second line of validity evidence. Pooled across all module-participant pairs ($n = 47$), the two measures correlated at $r_s = -.514$, $p < .001$ (see Table 5.8). (Hart et al. 1988) When participants reported higher understanding they also reported lower cognitive load, a convergence consistent with the expectation that genuine comprehension reduces processing effort. (Sweller 1988) The two instruments captured related but distinct facets of the learning experience: one tapped perceived mastery, the other tapped perceived demand. Their negative coupling supports the claim that both were measuring something real. M6 broke the pattern. At that module the correlation collapsed to $r_s = +.037$, $p = .937$, the only value in the dataset

indistinguishable from zero. One reading of that collapse is that cognitive overload at M6, the most demanding session on four of six TLX subscales, pushed working memory past the point where participants could judge their own understanding accurately; the self-assessment process itself becomes unreliable under saturation. (Sweller et al. 2019) That interpretation cannot be separated from two compounding factors: PyTorch installation failures consumed substantial time at M6, with four of seven challenge responses citing setup or hardware problems as their primary obstacle (Section 3.4.3), and end-of-day fatigue after roughly seven hours of continuous work likely inflated the TLX ratings beyond what content difficulty alone would produce.

What the assessment battery could not reach is equally important for RQ4. The proposal planned portfolio evaluation, artifact-based interviews (Brennan et al. 2012), and rubric-scored creative artefacts (Brookhart 2013); none were implemented. The knowledge test captures declarative and procedural understanding but says nothing about creative application, transfer to open-ended projects, or aesthetic judgement. The IDK reduction from 74% to 24% marks a metacognitive shift, not a skill, and P09's feedback form comment puts the gap in concrete terms: "2nd exercise is copy-paste; couldn't do it on my own." Completing a Create exercise and demonstrating the independent competence that a portfolio rubric would assess are not the same act, and the current instrument set cannot distinguish between them.

RQ4 gets a partial answer here, and only a partial one. This cycle validated a reliable, discriminating knowledge instrument and established convergent validity across two measurement approaches; both outcomes anchor future iterations. Creative-technical assessment, the dimension the research question was originally designed to address, remains an open design problem for the next DBR cycle. Section 6.7 catalogues the missing instruments, and Section 6.8 outlines the planned additions.

6.5. RQ5: Transfer and Application

Section C (Neural Networks) gained significantly: $W = 0.0$, $p = .031$, $d = 1.479$ (see Section 5.2). M5 was the only session that touched neural network content directly, and it covered a single-layer perceptron. The six Section C items asked about network architectures, training loops, and activation functions well beyond that single exposure. Participants had to carry what they picked up about threshold units into territory the workshop never taught. That pattern matches what Perkins et al. (1992) call near transfer: applying a learned procedure to a structurally adjacent problem without deliberate abstraction. The gain is consistent with that reading, though the instrument tested recognition and recall, not open-ended problem solving. Barnett et al. (2002) stress that even near transfer weakens as the gap between training content and test content widens; six items spanning three sub-topics push harder than one module can reliably anchor.

Section D (Generative AI) improved but fell short of significance: $g = 0.271$, $p = .062$ (see Table 5.3), and 56% of post-test responses were still "I Don't Know." Transfer to the

most abstract content domain was plainly incomplete. The broader IDK picture, though, looks different. The 50-point collapse in IDK rates, from 74% of responses before the workshop to 24% afterward (Section 5.2), concentrated in Sections A through C. What changed was not primarily skill; it was disposition. Participants who had declined to attempt items before the workshop now wrote answers, even wrong ones. That willingness to engage with unfamiliar material is closer to what Bransford et al. (1999) call preparation for future learning: the measure that matters is not whether learners answer correctly on the day but whether the experience leaves them better equipped to learn from whatever comes next. (Bransford et al. 1999, p. 69) Feedback form ratings drew the same line. C6 (foundational concepts connect to AI) averaged $M = 4.17$ ($SD = 0.98$, $n = 6$); C3 (confident applying concepts to own projects) sat at $M = 3.50$ ($SD = 1.22$). Participants saw the bridge. They did not yet trust themselves to cross it alone.

The qualitative record puts boundaries on where transfer held and where it broke (Section 5.6). CONN_EXPLICIT codes, marking responses in which a participant named a specific prior concept, clustered in M2 through M4 (12 of 16 total applications); arrays and kernels were the linking ideas cited most often. CONN_NONE codes (8 applications) spread across M2–M6, driven largely by two participants who consistently struggled to articulate connections, but the pattern intensified beyond the nine-module gap in the workshop sequence where no intermediate content bridged the jump. P05's feedback form comment captures the disposition that gap produced: "While I don't think I fully grasp the concepts from all the modules, this was an excellent introduction for further self-learning. I find gaps in the teaching materials challenging but in a positive way; now I can try to fill them myself." What that statement describes might be called dispositional transfer: not mastery, but a willingness to keep working toward it under cognitive strain.

Of the five research questions, RQ5 draws on the thinnest evidence. Near transfer within the technical domain is partly supported; far transfer and creative application were not tested. Covering the omitted Modules 7 through 9 and adding portfolio-based assessment (see Section 6.8) would be the minimum needed to give the question a fuller answer.

6.6. Connections to Existing Literature

Cognitive load theory predicted most of what happened. The experience-load correlation ($r_s = -.857$, $p = .007$) slots into the expertise reversal literature without modification. Tetzlaff et al. (2025) reported meta-analytic effect sizes of $d = 0.505$ for novice benefit under high instructional assistance and $d = -0.428$ for expert disadvantage under the same conditions; the present sample split along the same axis, with experienced participants (P01, P02, P03) running TLX scores roughly half those of the novices. What CLT did not predict was the stability of knowledge gains across the experience range. The non-significant experience-gain correlation ($r_s = +.204$, $p = .629$) means the curriculum served both groups on the outcome that matters most, even though their processing cost differed sharply. Bjork et al. would

call the extra novice burden a “desirable difficulty” (Bjork et al. 2011, p. 58): harder, but not unproductive. Sweller et al. (2019) would classify the excess load as a mix of intrinsic rather than extraneous, since the higher mental demand mapped onto content-relevant processing (mathematical notation, conceptual abstraction) rather than onto instructional clutter.

The module-level TLX trajectory ties to a second CLT mechanism. Mayer et al. (2003) listed segmentation among his nine load-reduction strategies; the workshop split the curriculum into six blocks, each capped by an exit ticket. The Friedman test on overall workload ($\chi^2(5) = 15.795$, $p = .008$, $W = 0.351$) shows that the segments did not carry equal load. Segmentation worked, but not because every segment was easy. It worked because the variation became visible and assessable, module by module, rather than buried in a single end-of-day score.

Dual coding and constructionism overlap more in practice than the literature review acknowledged. Every module paired code with a visual artifact: a pixel array, a cellular automaton, a fractal, a perceptron boundary, a noise schedule. That pairing satisfies Mayer (2014)’s multimedia principle (verbal and pictorial channels carrying complementary information) and Papert (1980)’s constructionist claim (learners build shareable objects). The Hands-On Discovery (HOD) modules put both to work at once. Participants wrote code, watched the output, and revised. Brennan (2015) drew the distinction: constructionist teaching means building *with* technology, not studying *about* it (Brennan 2015, p. 291). The Hands-On Discovery (HOD) workload advantage ($d = 0.699$ on overall TLX, $d = 1.100$ on mental demand) points to a measurable payoff from that coupling. M3’s anomalous success within Conceptual Deep-Dive (CDD) reinforces the point from the other direction. That session opened with a visual anchor before introducing notation, borrowing Hands-On Discovery (HOD)’s core mechanism under a different label. The visual artifact did the load reduction. The framework label did not.

AI literacy frameworks supplied the sequencing logic behind the curriculum’s structure. Long et al. (2020) and Ng et al. (2021) treat AI literacy as a staged progression rather than a single threshold, and the SAIL model from MacCallum et al. (2024) orders those stages into a scaffolded progression. The workshop tested the first two levels (Know and Use). Knowledge gains at those levels were strong. Section D (Generative AI), which maps to the higher Evaluate and Create levels, produced the weakest outcome ($g = 0.271$, 56% residual IDK). Every staged framework predicts exactly this: higher-order competencies depend on lower-order mastery, and a compressed schedule cannot supply the intermediate steps. The frameworks are right about the sequence. The data are right about the cost of skipping it.

Design-based research sets the boundary around what the study can claim. Cobb et al. called first-cycle outputs “humble theories” (Cobb et al. 2003, p. 9): local conjectures grounded in data but not yet tested across settings, populations, or iterations. Three conjectures survive this cycle. First, visual-first scaffolding lowers cognitive load relative to concept-first instruction. Second, prior experience modulates that load without suppressing learning gains. Third, staged AI literacy curricula break when intermediate stages are removed. Each conjecture is

specific enough to test in a second cycle and broad enough to carry beyond this particular curriculum. Collins et al. (2004) set the standard for DBR: refine the artifact and advance the theory in the same motion. The artifact (the curriculum) now has six modules' worth of empirical feedback. The theory (progressive creative AI pedagogy) has three grounded conjectures and a list of what the next cycle must address.

6.7. Limitations

Nine participants, no control group, and a single seven-hour session cap the conclusions this work can support. Each constraint is catalogued below, with back-references to where it first appeared.

Sample size and design. Every inferential test in Chapter 5 ran non-parametric at $n = 9$ ($n = 8$ for paired knowledge comparisons). Even the strongest possible Wilcoxon outcome could not survive Bonferroni correction on pairwise module contrasts (Section 5.3). Power was low. Without a control or comparison group, alternative explanations for the pre-to-post gains, including testing effects, maturation over the session day, and the social pressure of a group setting, remain uncontrolled. (Shadish et al. 2002) Recruitment drew on personal and university networks (Section 3.5), and everyone who responded was accepted. Volunteers who sign up for a coding workshop are not the same population the curriculum eventually needs to reach; that self-selection pressure likely pushed both engagement and outcomes upward.

Temporal scope. What the full curriculum spreads across weeks was compressed into roughly seven hours of hands-on work. No delayed post-test was administered. Follow-up interviews had been scheduled for two to three weeks after the session, but participant response was insufficient and the interviews were dropped (Section 3.2). All learning claims in the thesis rest on same-day data. Whether the gains persisted, decayed, or consolidated afterward is an open question.

Curriculum coverage. Six of fifteen modules reached the workshop floor. The other nine sat between partial drafts and structural placeholders (Section 3.3). RQ3, which targets TouchDesigner integration, could not be tested because the three relevant modules (10, 11, 13) were among the nine; Section 6.3 handled the question through design rationale alone. The omission also broke the scaffolding chain for generative AI content: Section D of the knowledge test returned the weakest gains ($g = 0.271$, $p = .062$) and kept 56% of responses at "I Don't Know" post-test (Section 6.1). That outcome reflects the compressed sequence at least as much as it reflects the content.

Instrument gaps. Three planned tools never made it to the room. The Creative Problem-Solving Efficacy Scale was cut for time, think-aloud protocols were infeasible with one facilitator running the session, and expert curriculum review was deferred to a later cycle. Creative self-efficacy therefore went unmeasured, real-time cognitive processes went unobserved, and content validity rested on the researcher's judgement alone. The knowledge test checked recognition and recall; portfolio evaluation, rubric-scored creative artefacts, and peer assessment were proposed but not built (Section 6.4). No far-transfer instrument existed. RQ5 accordingly draws on the thinnest evidence base of the five questions (Section 6.5).

Researcher role and data quality. The researcher designed the curriculum, ran the workshop, and was the sole coder for all 204 qualitative code applications. No second analyst checked the assignments; no inter-rater statistic was computed (Section 3.7). Participants knew the facilitator had built the material they were working through, which opens the door to demand characteristics. (Orne 1962) Structured observation notes were kept during the session but not transcribed or systematically coded before the submission deadline. Their contribution was zero. The post-workshop evaluation form was unplanned. Seven of nine participants returned it (P07 and P08 did not); those responses appear throughout the discussion as supplementary descriptive evidence, not primary data (Section 5.7).

Outlier handling. P08 rated four of six TLX subscales at the scale floor across every module, a pattern consistent with satisficing. Every key test was re-run without P08; no result changed direction or crossed a significance boundary (Section 3.7). P08 stayed in the dataset. The flag is recorded here because satisficing cannot be ruled out at $n = 9$, even though the numbers held.

These constraints place the study within what Cobb et al. (2003) called first-cycle DBR: grounded conjectures, not portable theory. A second iteration would need a larger and more diverse sample, delayed assessment, the missing instruments, and broader module coverage. Section 6.8 sets out that plan.

6.8. Implications for Practice

C2 on the post-workshop evaluation form ("Visual examples helped me understand") scored $M = 4.83$ ($SD = 0.41$, $n = 6$), the highest-variance learning-outcome item in the battery (C1 sat at ceiling, $M = 5.00$, $SD = 0.00$) (Section 5.7). That number, together with the workload and qualitative evidence reported in Section 6.1 through Section 6.5, points toward recommendations for five groups of practitioners. Each recommendation below traces to a specific finding.

Curriculum designers. Visual-first scaffolding belongs at the front of any creative coding sequence. Hands-On Discovery (HOD) modules carried lower overall workload than

Conceptual Deep-Dive (CDD) modules ($d = 0.699$) and substantially lower mental demand ($d = 1.100$, $p = .016$; Table 5.6), while ENG_CREATIVE codes clustered almost exclusively in Hands-On Discovery (HOD) sessions (4 of 5 applications; Section 5.6). Open each content domain with a module whose first action produces a visible artifact; defer formal notation to a follow-up session. M3's success within Conceptual Deep-Dive (CDD) shows that even concept-heavy content benefits when a perceptual anchor comes before the abstraction (Mayer 2014) (Section 6.1).

Facilitators. The experience-load correlation reported in Section 6.2 ($r_s = -.857$, $p = .007$) sat next to a flat experience-gain relationship ($r_s = +.204$, $p = .629$). Novices and experienced coders both improved, but their processing cost differed sharply. A single-track delivery cannot serve both. Five of seven feedback respondents rated the pace as slightly fast or too fast, and P05 wrote, "Could break down the modules more for beginners" (Section 5.7). P01 offered a concrete fix: "Beginners should use Google Colab instead of installing locally." Setup friction is one target; adaptive pacing is the other. Optional extension tasks for advanced participants, slower walk-throughs for novices, and experience-based grouping where class sizes permit would address the split (van de Pol et al. 2010). Conceptual Deep-Dive (CDD) modules need the heaviest scaffolding; the mental demand gap in Section 6.1 makes that case on its own.

Platform developers. P09 called the Create exercises "copy-paste; couldn't do it on my own" and flagged animated GIFs as "too fast." P06 asked to "cover step by step." P09 also requested "a one-liner overview at the start of each topic." CHAL_VISUAL_GAP appeared in M2 when two participants independently noted missing intermediate output in the cellular automata exercise (Section 5.6). The common thread across these responses is visibility: slower or pausable demonstrations, scaffolded tasks that require genuine modification rather than verbatim reproduction, and brief orienting summaries before each module begins (Mayer et al. 2003). At M6 the Q4-TLX correlation collapsed to $r_s = +.037$ (Section 6.4). Once cognitive saturation sets in, learners stop self-regulating. (Sweller et al. 2019) Platform-level pause points or progress checks could catch that threshold before participants cross it.

Researchers. At $n = 9$, every inferential test ran non-parametric. Wilcoxon, Friedman, and Spearman tests, paired with rank-biserial effect sizes, handled the analysis without distributional assumptions and still picked up a large knowledge effect ($d = 1.615$), a strong experience-load association ($r_s = -.857$), and significant load variation ($W = 0.351$; Chapter 5). The toolkit is replicable for small-sample DBR pilots. Layering a convergent parallel design on top (Section 5.7) added explanatory depth: qualitative codes unpacked patterns that significance tests could flag but not interpret, and feedback form responses supplied participant voice that exit tickets, framed within the session, could not.

Future iterations. Scope reduction is the first priority. Six modules in roughly seven hours pushed five of seven feedback respondents past a comfortable pace (Q6; Section 5.7). P06 put it plainly: “Maybe more time, more explorations.” The next cycle should either halve the module count per session or spread the workshop across two days. Whichever path is taken, the omitted intermediate modules (7 through 9) must be restored before generative AI content can be assessed against the full scaffolding logic; the Section D gap (Section 6.1) and the chain argument in Section 6.5 both depend on that restoration. Portfolio-based assessment, peer evaluation rubrics, and creative self-efficacy measurement, all absent from this cycle (Section 6.4), are planned for Cycle 2.

7. Conclusion

7.1. Summary of Findings

Nine participants worked through six modules of a creative coding curriculum on a single day in February 2026. The modules spanned pixel-level colour mixing, cellular automata, convolution kernels, fractal recursion, perceptron classification, and denoising diffusion. Two instructional frameworks alternated across the sequence: Hands-On Discovery, which opened each topic with a runnable visual artefact, and Conceptual Deep-Dive, which led with formal notation before code. Pre- and post-test scores, per-module NASA-TLX ratings, exit tickets, and a supplementary post-workshop evaluation form supplied the data. Five research questions guided the analysis; each is summarised here, with full interpretations in Chapter 6.

RQ1 asked what design patterns scaffold creative coding toward generative AI. Hands-On Discovery (HOD) modules carried lower workload than Conceptual Deep-Dive (CDD) modules on seven of eight NASA-TLX dimensions, and the mental demand gap was significant and large ($p = .016$, $d = 1.100$). Pre-to-post knowledge gains reached $W = 0.0$, $p = .008$, $d = 1.615$ across the full 24-item test. M3 (Convolution), a Conceptual Deep-Dive (CDD) session, broke the pattern by producing the highest single-module understanding rating ($M = 3.88$), evidence that concept-first instruction can work when paired with strong visual grounding. The answer to RQ1 is not that one framework outperforms the other universally; it is that content type determines which approach imposes less cognitive cost.

RQ2 targeted cognitive load management. The six-module load trajectory varied significantly ($\chi^2(5) = 15.795$, $p = .008$, $W = 0.351$), and mental demand, rather than time pressure or physical effort, drove most of that variation. Prior programming experience turned out to be the single strongest predictor in the dataset ($r_s = -.857$, $p = .007$, $n = 8$). That correlation held after excluding the lowest-variance participant. Experience did not predict learning gains ($r_s = +.204$, $p = .629$), meaning novices and experienced coders improved at comparable rates but at different processing costs.

RQ3, which targeted TouchDesigner integration, went untested because no TouchDesigner modules reached the workshop floor. The three relevant modules (10, 11, and 13) were among the nine that remained at placeholder status. The question was addressed through design rationale alone (Section 6.3), drawing on transfer-of-learning theory and the curriculum's structural logic rather than participant data.

RQ4 asked how learning outcomes in creative AI education can be assessed. The 24-item knowledge test returned a post-test KR-20 of 0.913, and exit ticket self-ratings converged

with TLX scores at $r_s = -.514$ ($p < .001$, $n = 47$ pooled pairs). Those numbers support instrument reliability and convergent validity for the measures that were deployed, though that convergence collapsed at M6 (DDPM), where cognitive overload, compounded by installation difficulties and session-end fatigue (Section 3.4.3), appeared to disrupt self-assessment ($r_s = +.037$, $p = .937$). Portfolio-based and creative assessment remain untested.

RQ5 addressed transfer. Near transfer showed up in Section C of the test ($p = .031$, $d = 1.479$), while Section D (generative AI) improved modestly ($p = .062$, $d = 0.689$) and kept 56% of responses at "I Don't Know" post-test. "I Don't Know" selections across the full test dropped from 74% to 24%, a shift that suggests growing confidence even where mastery did not follow. Near transfer within the technical domain found partial support, but far transfer and creative application were not tested in this cycle.

7.2. Contributions

The study makes contributions in three areas: theory, practice, and method.

7.2.1. Theoretical Contributions

Creative coding curricula that bridge visual computation and generative AI had not been tested through the lens of Cognitive Load Theory (Sweller et al. 2019). The $r_s = -.857$ experience-load correlation reported here adds a data point to that gap: learners with richer schemas processed the same material at markedly lower cognitive cost, exactly as element interactivity theory predicts (Sweller et al. 1998). Matching that finding with the non-significant experience-gain association ($r_s = +.204$, $p = .629$) sharpens the picture. Prior knowledge reduced processing cost without capping learning; both novices and experienced coders improved at comparable rates. The HOD-versus-CDD comparison adds a design-level corollary: visual-first scaffolding lowers the entry barrier for topics where spatial output is available, while concept-first instruction remains necessary for content that cannot be grounded in a runnable image before formalism is introduced. M3 (Convolution) sat inside Conceptual Deep-Dive (CDD) yet produced the highest single-module understanding rating ($M = 3.88$), a result that points to content-framework fit rather than blanket superiority of either approach.

7.2.2. Practical Contributions

The full fifteen-module curriculum is deployed as an open-access website and backed by a public GitHub repository (Figure 1.2). Six modules were tested under workshop conditions; of the remaining nine, one had most of its exercises published, five carried partial content at varying stages of completion, and three remained at structural-placeholder status with no published exercises (Chapter C). Practitioners in creative computing, media arts, or

introductory AI courses can adopt or adapt the material without licensing restrictions. The assessment instruments, a 24-item knowledge test, a per-module NASA-TLX battery, an exit ticket protocol with four open-ended prompts and one Likert item, and a supplementary feedback form, are documented in Chapter A and can be reused in comparable settings. The workshop format itself, six modules across roughly seven hours with interleaved surveys, proved feasible for a solo facilitator, though the pace data reported in Section 5.7 suggest that halving the module count per session or spreading the material across two days would better serve mixed-experience groups. Five of seven feedback respondents rated the pace as slightly fast or too fast.

7.2.3. Methodological Contributions

Design-Based Research has historically emphasised qualitative methods (Anderson et al. 2012), and small-sample studies rarely attempt inferential statistics. This study shows that non-parametric inferential tests (Wilcoxon, Friedman, Spearman) paired with rank-biserial effect sizes can extract stable patterns from $n = 9$ without distributional assumptions. Every key result was re-run with and without the lowest-variance participant (P08); no finding changed direction or crossed a significance boundary (Section 3.7). The convergent parallel mixed-methods structure (Creswell et al. 2017) layered quantitative workload and knowledge data with thematic analysis of 204 coded exit ticket segments, letting each strand check the other. Per-module TLX administration, rather than the more common single post-task measurement, turned the cognitive load instrument into a formative design tool: the module-level trajectory exposed where the curriculum worked and where it broke down, information that a session-level average would have hidden. The NASA-TLX dataset itself had no gaps: every participant rated every module, a complete 9×6 matrix. Across all instruments, 127 of 135 data cells were filled (94.1%), an uncommonly high figure for a single-day workshop with voluntary instruments. That level of completeness, together with the 204 qualitative code applications drawn from six rounds of exit tickets, gave the convergent analysis a grain size fine enough to distinguish module-level patterns that a coarser design would have missed entirely.

7.3. Future Work

Several limitations outlined in Section 6.7 point directly to second-cycle priorities.

Curriculum completion. Nine of fifteen modules went untested during the workshop. Three of those nine had no published content at all (Modules 7, 13, and 14); the remainder ranged from minimal to substantial coverage (Chapter C). Completing Modules 7 through 9, the intermediate bridge between classical algorithms and neural networks, is the single highest-priority task. Without them, the scaffolding logic behind Sections C and D of the knowledge

test cannot be evaluated against its intended prerequisite chain. The Section D result, $g = 0.271$ with a 56% residual "I Don't Know" rate, may reflect compressed sequencing as much as content difficulty. Participants jumped from fractal geometry directly to perceptrons and diffusion; the missing modules on gradient descent, loss functions, and basic neural network architecture were supposed to bridge that gap.

Expanded evaluation. A multi-session workshop spread across two or more days, with 20 to 30 participants drawn from varied institutional backgrounds, would address the sample-size constraint and permit parametric analysis. At $n = 9$, every inferential test in this study ran non-parametric. The results held, but the confidence intervals were wide and the statistical power to detect small effects was low. Delayed post-testing at two and four weeks would show whether gains persist or decay; the current dataset is a same-day snapshot only (Section 6.7). Adding the Creative Problem-Solving Efficacy Scale, portfolio rubrics scored by independent raters, and a second qualitative coder would fill the instrument gaps documented in Section 6.4 and raise inter-rater reliability from its current single-coder baseline. Think-aloud protocols, cut from the current cycle for logistical reasons, would add real-time process data that exit ticket reflections, collected after the fact, cannot supply.

TouchDesigner integration. Modules 10, 11, and 13 target real-time visual programming through TouchDesigner. Two of seven feedback respondents spontaneously requested TouchDesigner content on the post-workshop form (Section 5.7), suggesting participant interest exists. Piloting those modules would convert RQ3 from a design argument into an empirical one. Transfer measurement between the Python-based modules and the node-based TouchDesigner environment would supply the strongest available test of whether foundational concepts generalise across toolchains.

Longitudinal and multi-site replication. A single-day snapshot cannot establish retention, far transfer, or curricular sustainability. Tracking a cohort across a full semester, with periodic knowledge checks and portfolio milestones, would test the curriculum under conditions closer to standard university deployment. Replication at a second institution would separate the curriculum's effect from facilitator-specific or site-specific factors. The researcher's dual role as designer and facilitator (Section 6.7) introduced demand characteristics that only an independent replication can control for.

7.4. Closing Reflection

The introduction to this thesis opened with a single coloured pixel on screen and asked whether a structured curriculum could carry a learner from that pixel to a generative model. After one Design-Based Research cycle, the honest answer is: partway. Nine participants worked through six modules across a seven-hour workshop day. Every one of them scored

higher on the post-test than on the pre-test. The curriculum moved them from RGB arrays through cellular automata, convolution, and fractal recursion to a perceptron classifier and a simplified denoising diffusion model. That progression held together; the load data and the knowledge gains backed each other up, and the exit tickets added texture to both. Where it strained was at the far end: Section D of the post-test kept 56% of responses at “I Don’t Know,” and five of seven feedback respondents called the pace slightly fast or too fast. Two modules and roughly fifty minutes of instruction cannot stand in for intermediate content that the curriculum’s middle tier was supposed to supply: gradient descent, loss landscapes, and basic network architecture all sat in modules that never reached the workshop floor. Of those nine untested modules, six had partial implementations at various stages of development. The remaining three were structural placeholders only. Bridging classical image algorithms to neural generative models needs that middle tier built out and tested before any stronger claim about end-to-end scaffolding is warranted.

The researcher designed the curriculum, wrote the instruments, ran the workshop, and assembled the analysis pipeline solo. That workload turned scope constraints from a planning abstraction into something concrete. At the proposal stage, fifteen modules with 199 exercises sounded feasible. One development cycle produced 51 published exercises spread across all fifteen modules, but only six modules reached the level of completeness required for live workshop testing. That ratio, six workshop-ready modules out of fifteen, matters for anyone attempting solo curriculum development under a comparable timeline, and it is worth reporting as a finding in its own right.

The single pattern from the workshop day that stays clearest is not a p-value. Experienced participants carried markedly lower cognitive cost through the same material than novices did, yet both groups improved at comparable rates. The curriculum asked a different price depending on who sat in front of it, but it did not cap who could learn. If a second iteration fills in the missing modules, spreads the content across more than one day, and draws a larger cohort, the path from a coloured pixel to a generative model does not need to be shorter. It needs to be complete.

References

Adhikari, Arun (2023). “Thinking beyond Chatbots’ Threat to Education: Visualizations to Elucidate the Writing or Coding Process”. In: *Education Sciences* 13.9, p. 922. DOI: 10.3390/educsci13090922.

Alves, Nathalia da Cruz, Christiane Gresse von Wangenheim, and Fernando Silva Pacheco (2021). “Assessing Product Creativity in Computing Education: A Systematic Mapping Study”. In: *Informatics in Education* 20.4, pp. 487–515. DOI: 10.15388/infedu.2021.21.

Amankwah-Amoah, Joseph et al. (2024). “The Impending Disruption of Creative Industries by Generative AI: Opportunities, Challenges, and Research Agenda”. In: *International Journal of Information Management* 79, p. 102759. DOI: 10.1016/j.ijinfomgt.2024.102759.

Anderson, Terry and Julie Shattuck (2012). “Design-Based Research: A Decade of Progress in Education Research?” In: *Educational Researcher* 41.1, pp. 16–25. DOI: 10.3102/0013189X11428813.

Anthropic (2026). *Claude Opus 4.6 [Large Language Model]*. Retrieved January–March 2026. Used for curriculum design brainstorming, thesis structuring, instrument design assistance (feedback and observation forms), pre-/post-test item validation, figure and table generation in LaTeX (including data-visualisation charts), data-processing script debugging, statistical formatting and interpretation verification, and proofreading. All outputs were critically reviewed, adapted, and verified by the author. URL: https://claude.ai.

Appiah-Twumasi, Eric (2024). “Scaffolding as a Cognitive Load Reduction Strategy for Teaching Atomic and Nuclear Physics”. In: *Momentum: Physics Education Journal* 8.2, pp. 194–209. DOI: 10.21067/mpej.v8i2.9580.

Aristizábal Zapata, Jhon Hader, Mariana Alejandra Posada, and Pascual D. Diago (2024). “Design and Validation of a Computational Thinking Test for Children (CTTC)”. In: *Multimodal Technologies and Interaction* 8.5, p. 39. DOI: 10.3390/mti8050039.

Baldwin, Lynne P. and Jasna Kuljis (2000). “Visualisation Techniques for Learning and Teaching Programming”. In: *Journal of Computing and Information Technology* 8.4, pp. 285–291. DOI: 10.2498/cit.2000.04.03.

Barnett, Susan M. and Stephen J. Ceci (2002). “When and Where Do We Apply What We Learn? A Taxonomy for Far Transfer”. In: *Psychological Bulletin* 128.4, pp. 612–637. DOI: 10.1037/0033-2909.128.4.612.

Basu, Satabdi (2019). “Using Rubrics Integrating Design and Coding to Assess Middle School Students’ Open-Ended Block-Based Programming Projects”. In: *Proceedings of the 50th ACM Technical Symposium on Computer Science Education (SIGCSE ’19)*. ACM, pp. 1211–1217. DOI: 10.1145/3287324.3287412.

Bäuerle, Alex et al. (2022). “exploRNN: Teaching Recurrent Neural Networks through Visual Exploration”. In: *The Visual Computer* 38, pp. 4073–4088. DOI: 10.1007/s00371-022-02593-0.

Bjork, Robert A. and Elizabeth L. Bjork (2011). “Making Things Hard on Yourself, but in a Good Way: Creating Desirable Difficulties to Enhance Learning”. In: *Psychology and the Real World: Essays Illustrating Fundamental Contributions to Society*. Ed. by Morton A. Gernsbacher et al. New York: Worth Publishers, pp. 56–64.

Blake-West, Julia, Mohammad Alrawashdeh, and Marina Umaschi Bers (2024). “Validating a Creative Coding Rubric through Expressive Activities for Elementary Grades”. In: *Journal of Research on Technology in Education*. DOI: 10.1080/15391523.2024.2313936.

Bransford, John D. and Daniel L. Schwartz (1999). “Rethinking Transfer: A Simple Proposal with Multiple Implications”. In: *Review of Research in Education* 24, pp. 61–100. DOI: 10.2307/1167267.

Braun, Virginia and Victoria Clarke (2006). “Using Thematic Analysis in Psychology”. In: *Qualitative Research in Psychology* 3.2, pp. 77–101. DOI: 10.1191/1478088706qp063oa.

Braun, Virginia and Victoria Clarke (2019). “Reflecting on Reflexive Thematic Analysis”. In: *Qualitative Research in Sport, Exercise and Health* 11.4, pp. 589–597. DOI: 10.1080/2159676X.2019.1628806.

Brennan, Karen (2015). “Beyond Technocentrism: Supporting Constructionism in the Classroom”. In: *Constructivist Foundations* 10.3, pp. 289–296.

Brennan, Karen and Mitchel Resnick (2012). “New Frameworks for Studying and Assessing the Development of Computational Thinking”. In: *Proceedings of the Annual Meeting of the American Educational Research Association*. Vancouver, Canada.

Brookhart, Susan M. (2013). *How to Create and Use Rubrics for Formative Assessment and Grading*. ASCD.

Brown, Ann L. (1992). “Design Experiments: Theoretical and Methodological Challenges in Creating Complex Interventions in Classroom Settings”. In: *The Journal of the Learning Sciences* 2.2, pp. 141–178.

Bruner, Jerome S. (1960). *The Process of Education*. Harvard University Press.

Çakıroğlu, Ünal et al. (2018). “Exploring Perceived Cognitive Load in Learning Programming via Scratch”. In: *Research in Learning Technology* 26. DOI: 10.25304/rlt.v26.1888.

Carney, Michelle et al. (2020). “Teachable Machine: Approachable Web-Based Tool for Exploring Machine Learning Classification”. In: *Extended Abstracts of the 2020 CHI Conference on Human Factors in Computing Systems*. ACM, pp. 1–8. DOI: 10.1145/3334480.3382839.

Cassidy, Kathleen (2008). “Promoting Creativity in Computing via Portfolio Assessment”. In: *Blended Learning in Practice* 1.1. DOI: 10.21913/ATNA.V1I1.281.

Chen, Junyi et al. (2024). “MindScratch: A Visual Programming Support Tool for Classroom Learning Based on Multimodal Generative AI”. In: *International Journal of Human-Computer Interaction*. DOI: 10.1080/10447318.2024.2443297.

Chung, Wilbert (2025). “Programming Learning Platform with Visual Aids”. In: *Journal of International Conference Proceedings* 8.1. DOI: 10.32535/jicp.v8i1.4002.

Cobb, Paul et al. (2003). “Design Experiments in Educational Research”. In: *Educational Researcher* 32.1, pp. 9–13. DOI: 10.3102/0013189X032001009.

Cohen, Jacob (1988). *Statistical Power Analysis for the Behavioral Sciences*. 2nd ed. Lawrence Erlbaum Associates.

Collins, Allan, Diana Joseph, and Katerine Bielaczyc (2004). “Design Research: Theoretical and Methodological Issues”. In: *The Journal of the Learning Sciences* 13.1, pp. 15–42. DOI: 10.1207/s15327809jls1301_2.

Corrales-Álvarez, Lilia J., Adrián Muñoz-Muñoz, and Sergio A. Cardona-Torres (2025). “Computational Thinking in the University Context: A Literature Review of Assessment Instruments”. In: *TecnoLógicas* 28.61. DOI: 10.22430/22565337.3394.

Creswell, John W. and Vicki L. Plano Clark (2017). *Designing and Conducting Mixed Methods Research*. 3rd ed. SAGE Publications.

Derivative (2023). *TouchDesigner Curriculum*. URL: https://learn.derivative.ca/.

Design-Based Research Collective (2003). “Design-Based Research: An Emerging Paradigm for Educational Inquiry”. In: *Educational Researcher* 32.1, pp. 5–8. DOI: 10.3102/0013189X032001005.

Domínguez-Gómez, Pedro and Fánel Celis (2024). “Creative Programming in Architecture: A Computational Thinking Approach”. In: *Informatics in Education*. DOI: 10.15388/infedu.2024.13.

Dunn, Olive Jean (1961). “Multiple Comparisons among Means”. In: *Journal of the American Statistical Association* 56.293, pp. 52–64. DOI: 10.1080/01621459.1961.10482090.

Ebel, Robert L. and David A. Frisbie (1991). *Essentials of Educational Measurement*. 5th ed. Englewood Cliffs, NJ: Prentice Hall.

Elicit (2026). *Elicit: The AI Research Assistant*. Retrieved January–February 2026. Used during the literature review phase to identify relevant papers for the five research questions. All results were verified against original sources. URL: https://elicit.com.

Endres, Tino et al. (2023). “Can Prior Knowledge Increase Task Complexity? Cases in Which Higher Prior Knowledge Leads to Higher Intrinsic Cognitive Load”. In: *British Journal of Educational Psychology* 93.Suppl. 2, pp. 305–317. DOI: 10.1111/bjep.12612.

Ezeamuzie, Ndudi O. (2022). “Abstractive-Based Programming Approach to Computational Thinking: Discover, Extract, Create, and Assemble”. In: *Journal of Educational Computing Research* 60.5, pp. 1262–1289. DOI: 10.1177/07356331221081753.

Gardner, Martin (1970). “Mathematical Games: The Fantastic Combinations of John Conway’s New Solitaire Game “Life””. In: *Scientific American* 223.4, pp. 120–123.

Garner, Stuart (2002). “Reducing the Cognitive Load on Novice Programmers”. In: *Proceedings of ED-MEDIA 2002: World Conference on Educational Multimedia, Hypermedia & Telecommunications*. ERIC ED 477 013. AACE.

Gkintoni, Evgenia et al. (2025). “Challenging Cognitive Load Theory: The Role of Educational Neuroscience and Artificial Intelligence in Redefining Learning Efficacy”. In: *Brain Sciences* 15.2, p. 203. DOI: 10.3390/brainsci15020203.

Gonzalez, Aitor Arrieta, Lars Klevgaard, and Kjetil Raaen (2024). “Students’ Visualisations of Programming Concepts: An Exploratory Analysis”. In: *Proceedings of the Norwegian ICT Conference for Research and Education (NIKT 2024)*. DOI: 10.18261/nikt.2024.1.

Goodfellow, Ian et al. (2014). “Generative Adversarial Nets”. In: *Advances in Neural Information Processing Systems*. Vol. 27, pp. 2672–2680.

Gouldthorpe, Jessica L. and Glenn D. Israel (2013). “Capturing Change: Comparing Pretest-Posttest and Retrospective Evaluation Methods”. In: *EDIS* WC135. DOI: 10.32473/EDIS-WC135-2013.

Grover, Shuchi and Roy Pea (2013). “Computational Thinking in K–12: A Review of the State of the Field”. In: *Educational Researcher* 42.1, pp. 38–43. DOI: 10.3102/0013189X12463051.

Guggemos, Josef, Sabine Seufert, and Marcos Román-González (2019). “Measuring Computational Thinking – Adapting a Performance Test and a Self-Assessment Instrument for German-Speaking Countries”. In: *Proceedings of the International Conference on Cognition and Exploratory Learning in the Digital Age (CELDA 2019)*. IADIS, pp. 199–206.

Hafezi Fard, Mahsa et al. (2024). “The Effect of Prior Programming Knowledge on Memory Efficiency When Learning a New Language”. In: *Peer-Reviewed Abstracts of the 31st International Conference on Neural Information Processing (ICONIP 2024)*. Springer.

Hake, Richard R. (1998). “Interactive-Engagement versus Traditional Methods: A Six-Thousand-Student Survey of Mechanics Test Data for Introductory Physics Courses”. In: *American Journal of Physics* 66.1, pp. 64–74.

Harel, Idit and Seymour Papert, eds. (1991). *Constructionism: Research Reports and Essays, 1985–1990*. Ablex Publishing.

Hart, Sandra G. (2006). “NASA-Task Load Index (NASA-TLX); 20 Years Later”. In: *Proceedings of the Human Factors and Ergonomics Society Annual Meeting* 50.9, pp. 904–908. DOI: 10.1177/154193120605000909.

Hart, Sandra G. and Lowell E. Staveland (1988). “Development of NASA-TLX (Task Load Index): Results of Empirical and Theoretical Research”. In: *Human Mental Workload*. Ed. by Peter A. Hancock and Najmedin Meshkati. North-Holland, pp. 139–183.

Hmelo-Silver, Cindy E., Ravit Golan Duncan, and Clark A. Chinn (2007). “Scaffolding and Achievement in Problem-Based and Inquiry Learning: A Response to Kirschner, Sweller, and Clark (2006)”. In: *Educational Psychologist* 42.2, pp. 99–107.

Ho, Jonathan, Ajay Jain, and Pieter Abbeel (2020). “Denoising Diffusion Probabilistic Models”. In: *Advances in Neural Information Processing Systems*. Vol. 33, pp. 6840–6851.

Hoban, Garry, Wendy Nielsen, and Celina Carceller (2010). “Articulating Constructionism: Learning Science through Designing and Making “Slowmations””. In: *Research in Science Education* 40.5, pp. 609–624.

Hölbl, Marko and Lili Nemec Zlatolas (2019). “The Impact of Students’ Pre-Knowledge on Learning Computer Programming”. In: *Proceedings of the 8th Symposium on Languages, Applications and Technologies (SQAMIA 2019)*. Vol. 2508. CEUR Workshop Proceedings.

Huang, Jinbin et al. (2023). “ConceptExplainer: Interactive Explanation for Deep Neural Networks from a Concept Perspective”. In: *IEEE Transactions on Visualization and Computer Graphics* 29.1, pp. 831–841. DOI: 10.1109/TVCG.2022.3209384.

Jayaratne, K. S. U., Anil Kumar Chaudhary, and John M. Diaz (2025). “Knowledge Testing Options in Pre-Test Post-Test Evaluation Design”. In: *Advancements in Agricultural Development* 6.3, pp. 95–107. DOI: 10.37433/aad.v6i3.659.

Johnson, Colin (2017). “Learning Basic Programming Concepts with Game Maker”. In: *International Journal of Computer Science Education in Schools* 1.2. DOI: 10.21585/ijcses.v1i2.5.

Jordan, Kari L. et al. (2018). “Short-Format Workshops Build Skills and Confidence for Researchers to Work with Data”. In: *Proceedings of the ASEE Annual Conference and Exposition*. ASEE. DOI: 10.18260/1-2--30960.

Kelleher, Caitlin and Randy Pausch (2005). “Lowering the Barriers to Programming: A Taxonomy of Programming Environments and Languages for Novice Programmers”. In: *ACM Computing Surveys* 37.2, pp. 83–137.

Kirchler, Matthias et al. (2021). “Explainability Requires Interactivity”. In: *arXiv preprint arXiv:2109.07869*. DOI: 10.48550/arXiv.2109.07869.

Kirschner, Paul A., John Sweller, and Richard E. Clark (2006). “Why Minimal Guidance during Instruction Does Not Work: An Analysis of the Failure of Constructivist, Discovery, Problem-Based, Experiential, and Inquiry-Based Teaching”. In: *Educational Psychologist* 41.2, pp. 75–86.

Koc-Januchta, Marta M. et al. (2022). ““Connecting Concepts Helps Put Main Ideas Together”: Cognitive Load and Usability in Learning Biology with an AI-enriched Textbook”. In: *International Journal of Educational Technology in Higher Education* 19.1, p. 11. DOI: 10.1186/s41239-021-00317-3.

Lobo, Ruben et al. (2025). “Balanced Creative Coding for Motivation and Learning Transfer”. In: *Proceedings of the 56th ACM Technical Symposium on Computer Science Education (SIGCSE ’25)*. ACM. DOI: 10.1145/3641554.3701862.

Long, Duri and Brian Magerko (2020). “What Is AI Literacy? Competencies and Design Considerations”. In: *Proceedings of the 2020 CHI Conference on Human Factors in Computing Systems*, pp. 1–16. DOI: 10.1145/3313831.3376727.

Luxton-Reilly, Andrew et al. (2018). “Developing Assessments to Determine Mastery of Programming Fundamentals”. In: *Proceedings of the 2018 ITiCSE Working Group Reports*. ACM, pp. 47–69. DOI: 10.1145/3293881.3295779.

MacCallum, Kathryn, David Parsons, and Mahsa Mohaghegh (2024). “The Scaffolded AI Literacy (SAIL) Framework for Education”. In: *He Rourou* 1.1. DOI: 10.54474/herourou.v1i1.10835.

Marwan, Samiha et al. (2022). “Adaptive Immediate Feedback for Block-Based Programming: Design and Evaluation”. In: *IEEE Transactions on Learning Technologies* 15.6, pp. 684–697. DOI: 10.1109/TLT.2022.3225469.

Mayer, Richard E. (2014). “Cognitive Theory of Multimedia Learning”. In: *The Cambridge Handbook of Multimedia Learning*. Ed. by Richard E. Mayer. 2nd ed. Cambridge University Press, pp. 43–71.

Mayer, Richard E. and Roxana Moreno (2003). “Nine Ways to Reduce Cognitive Load in Multimedia Learning”. In: *Educational Psychologist* 38.1, pp. 43–52.

Mills, Kelly et al. (2024). *AI Literacy: A Framework to Understand, Evaluate, and Use Emerging Technology*. Report. Digital Promise. URL: https://digitalpromise.org/2024/06/18/ai-literacy-a-framework-to-understand-evaluate-and-use-emerging-technology/.

Minsky, Marvin and Seymour A. Papert (1969). *Perceptrons: An Introduction to Computational Geometry*. MIT Press.

Ng, Andrew (2021). *Machine Learning Specialization*. URL: https://www.coursera.org/specializations/machine-learning-introduction.

Ng, Davy Tsz Kit et al. (2021). “AI Literacy: Definition, Teaching, Evaluation and Ethical Issues”. In: *Proceedings of the Association for Information Science and Technology*. Vol. 58. 1. Wiley, pp. 504–509. DOI: 10.1002/pra2.487.

Nooijen, Christine C. A. van et al. (2024). “A Cognitive Load Theory Approach to Understanding Expert Scaffolding of Visual Problem-Solving Tasks: A Scoping Review”. In: *Educational Psychology Review* 36, p. 12. DOI: 10.1007/s10648-024-09848-3.

Nunnally, Jum C. (1978). *Psychometric Theory*. 2nd ed. New York: McGraw-Hill.

Ocampo, Jenny, Magaly Gaviria-Marin, and Luis Camacho (2024). “Systematic Review of Instruments to Assess Computational Thinking in Early Years of Schooling”. In: *Education Sciences* 14.10, p. 1124. DOI: 10.3390/educsci14101124.

Orne, Martin T. (1962). “On the Social Psychology of the Psychological Experiment: With Particular Reference to Demand Characteristics and Their Implications”. In: *American Psychologist* 17.11, pp. 776–783. DOI: 10.1037/h0043424.

Paivio, Allan (1986). *Mental Representations: A Dual Coding Approach*. Oxford University Press.

Papert, Seymour (1980). *Mindstorms: Children, Computers, and Powerful Ideas*. Basic Books.

Pearson, P. David and Margaret C. Gallagher (1983). “The Instruction of Reading Comprehension”. In: *Contemporary Educational Psychology* 8.3, pp. 317–344.

Peppler, Kylie A. and Yasmin B. Kafai (2005). “Creative Coding: Programming for Personal Expression”. In: *Proceedings of the 8th International Conference on Computer Supported Collaborative Learning (CSCL 2005)*. International Society of the Learning Sciences.

Perkins, David N. and Gavriel Salomon (1992). “Transfer of Learning”. In: *International Encyclopedia of Education*. 2nd ed. Pergamon Press.

Processing Foundation (2024). *Education Resources*. URL: https://processingfoundation.org/education.

Putra, P. O. H., Y. Yustina, and D. Suhendra (2022). “Development of Automated Assessment Tool to Measure Student Creativity in Computer Programming”. In: *Proceedings of the International Conference on Computer Science and Information Technology (ICCSIT 2022)*. DOI: 10.2991/978-94-6463-084-8_8.

Qamar, Noreen et al. (2025). “The Role of AI in Reducing Cognitive Overload in Complex Learning Environments”. In: *Review of Applied Management and Social Sciences* 8.2, pp. 1111–1127. DOI: 10.47067/ramss.v8i2.541.

Queiroz, Raul L. et al. (2020). “AI from Concrete to Abstract: Demystifying Artificial Intelligence to the General Public”. In: *AI & Society* 36, pp. 595–603. DOI: 10.1007/s00146-020-01058-5.

Reas, Casey and Ben Fry (2007). *Processing: A Programming Handbook for Visual Designers and Artists*. MIT Press.

Reeves, Thomas C. (2006). “Design Research from a Technology Perspective”. In: *Educational Design Research*. Ed. by Jan van den Akker et al. Routledge, pp. 52–66.

Resnick, Mitchel (2007). “Sowing the Seeds for a More Creative Society”. In: *Learning & Leading with Technology* 35.4, pp. 18–22.

Resnick, Mitchel et al. (2009). “Scratch: Programming for All”. In: *Communications of the ACM* 52.11, pp. 60–67.

Rohaeti, Eli and Arief Huda (2025). “Assessing Computational Thinking Skills of Science and Mathematics Upper-Secondary School Students”. In: *European Journal of Science and Mathematics Education* 13.2, pp. 91–109. DOI: 10.30935/scimath/17248.

Román-González, Marcos, Jesús Moreno-León, and Gregorio Robles (2017). “Complementary Tools for Computational Thinking Assessment”. In: *Proceedings of the International Conference on Computational Thinking Education (CTE 2017)*. The Education University of Hong Kong, pp. 79–84.

Román-González, Marcos, Jesús Moreno-León, and Gregorio Robles (2019). “Combining Assessment Tools for a Comprehensive Evaluation of Computational Thinking Interventions”. In: *Computational Thinking Education*. Springer, pp. 79–98. DOI: 10.1007/978-981-13-6528-7_6.

Rosenblatt, Frank (1958). “The perceptron: A probabilistic model for information storage and organization in the brain”. In: *Psychological Review* 65.6, pp. 386–408. DOI: 10.1037/h0042519.

Salleh, Siti Mastura, Zarina Shukur, and Hairulliza Mohamad Judi (2018). “Scaffolding Model for Efficient Programming Learning Based on Cognitive Load Theory”. In: *International Journal of Pure and Applied Mathematics* 118.7, pp. 77–83.

Sandoval, William A. and Philip Bell (2004). “Design-Based Research Methods for Studying Learning in Context: Introduction”. In: *Educational Psychologist* 39.4, pp. 199–201. DOI: 10.1207/s15326985ep3904_1.

Scherer, Ronny (2016). “Learning from the Past – The Need for Empirical Evidence on the Transfer Effects of Computer Programming Skills”. In: *Frontiers in Psychology* 7, p. 1390. DOI: 10.3389/fpsyg.2016.01390.

Shadish, William R., Thomas D. Cook, and Donald T. Campbell (2002). *Experimental and Quasi-Experimental Designs for Generalized Causal Inference*. Boston: Houghton Mifflin.

Shiffman, Daniel (2012). *The Nature of Code: Simulating Natural Systems with Processing*. Self-published. URL: http://natureofcode.com.

Siegel, Sidney (1956). *Nonparametric Statistics for the Behavioral Sciences*. New York: McGraw-Hill.

Soh, Leen-Kiat et al. (2015). “Learning through Computational Creativity”. In: *Communications of the ACM* 58.8, pp. 33–35. DOI: 10.1145/2699391.

Stachel, Jennifer et al. (2013). “Managing Cognitive Load in Introductory Programming Courses: A Cognitive Aware Scaffolding Tool”. In: *Journal of Integrated Design and Process Science* 17.1, pp. 37–54. DOI: 10.3233/jid-2013-0004.

Stager, Gary S. (2005). “Papertian Constructionism and the Design of Productive Contexts for Learning”. In: *EuroLogo 2005: Proceedings of the 10th European Logo Conference*, pp. 43–53.

Suh, Sangho and Pei An (2022). “Leveraging Generative Conversational AI to Develop a Creative Learning Environment for Computational Thinking”. In: *Companion Proceedings of the 27th International Conference on Intelligent User Interfaces*. ACM, pp. 73–76. DOI: 10.1145/3490100.3516473.

Sulmont, Elisa, Elizabeth Patitsas, and Jeremy R. Cooperstock (2019). “What Is Hard about Teaching Machine Learning to Non-Majors?” In: *Proceedings of the 50th ACM Technical Symposium on Computer Science Education*, pp. 412–418.

Sweller, John (1988). “Cognitive Load during Problem Solving: Effects on Learning”. In: *Cognitive Science* 12.2, pp. 257–285.

Sweller, John (2024). “Cognitive Load Theory and Individual Differences”. In: *Learning and Individual Differences* 109, p. 102365. DOI: 10.1016/j.lindif.2024.102365.

Sweller, John, Jeroen J. G. van Merriënboer, and Fred Paas (2019). “Cognitive Architecture and Instructional Design: 20 Years Later”. In: *Educational Psychology Review* 31.2, pp. 261–292. DOI: 10.1007/s10648-019-09465-5.

Sweller, John, Jeroen J. G. van Merriënboer, and Fred Paas (1998). “Cognitive Architecture and Instructional Design”. In: *Educational Psychology Review* 10.3, pp. 251–296.

Sydora, Christiane, Mehmet Er, and Michael Muehlebach (2026). “Teaching Machine Learning Fundamentals with LEGO Robotics”. In: *arXiv preprint*. DOI: 10.48550/arXiv.2601.19376.

Terroso, Luís and Mário Pinto (2022). “Programming for Non-Programmers: An Approach Using Creative Coding in Higher Education”. In: *Proceedings of the International Conference on Informatics Education and Practice (ICPEC 2022)*. Vol. 102. OASIcs. Schloss Dagstuhl, 13:1–13:10. DOI: 10.4230/OASIcs.ICPEC.2022.13.

Tetzlaff, Luisa et al. (2025). “A Cornerstone of Adaptivity: A Meta-Analysis of the Expertise Reversal Effect”. In: *Learning and Instruction* 98, p. 102142. DOI: 10.1016/j.learninstruc.2025.102142.

Tversky, Barbara (2011). “Visualizing Thought”. In: *Topics in Cognitive Science* 3.3, pp. 499–535. DOI: 10.1111/j.1756-8765.2010.01113.x.

Twabu, Khanyisile (2025). “Enhancing the Cognitive Load Theory and Multimedia Learning Framework with AI Insight”. In: *Discover Education* 4, p. 160. DOI: 10.1007/s44217-025-00592-6.

Van de Pol, Janneke, Monique Volman, and Jos Beishuizen (2010). “Scaffolding in Teacher–Student Interaction: A Decade of Research”. In: *Educational Psychology Review* 22.3, pp. 271–296.

Vygotsky, Lev S. (1978). *Mind in Society: The Development of Higher Psychological Processes*. Harvard University Press.

Wang, An-I and Rabail Tahir (2020a). “Design-Based Research on Integrating Learning Technology Tools into Higher Education Classes to Achieve Active Learning”. In: *Computers & Education* 154, p. 103935. DOI: 10.1016/j.compedu.2020.103935.

Wang, Junyu (2025). “Scaffolding Creativity: Integrating Generative AI Tools and Real-World Experiences in Business Education”. In: *Extended Abstracts of the 2025 CHI Conference on Human Factors in Computing Systems*. ACM. DOI: 10.1145/3706599.3720216.

Wang, Zijie J. et al. (2020b). “CNN 101: Interactive Visual Learning for Convolutional Neural Networks”. In: *Extended Abstracts of the 2020 CHI Conference on Human Factors in Computing Systems*. ACM, pp. 1–7. DOI: 10.1145/3334480.3382899.

Wang, Zijie J. et al. (2021). “CNN Explainer: Learning Convolutional Neural Networks with Interactive Visualization”. In: *IEEE Transactions on Visualization and Computer Graphics* 27.2, pp. 1396–1406. DOI: 10.1109/TVCG.2020.3030418.

Winter, Victor et al. (2019). “Using Visualization to Reduce the Cognitive Load of Threshold Concepts in Computer Programming”. In: *Proceedings of the 2019 IEEE Frontiers in Education Conference (FIE)*. IEEE. DOI: 10.1109/FIE43999.2019.9028498.

Wolf, Thomas (2012). *Brandenburg Gate [Photograph]*. Wikimedia Commons. CC BY-SA 3.0. URL: https://commons.wikimedia.org/wiki/File:Brandenburger_Tor_abends.jpg.

Wood, David, Jerome S. Bruner, and Gail Ross (1976). “The Role of Tutoring in Problem Solving”. In: *Journal of Child Psychology and Psychiatry* 17.2, pp. 89–100.

Wu, Ting-Ting, Adi Asmara, and Yueh-Min Huang (2023). “Tracking Visual Programming Language-Based Learning Progress for Computational Thinking Education”. In: *Sustainability* 15.3, p. 1983. DOI: 10.3390/su15031983.

Xu, Dianna et al. (2018). “Updating Introductory Computer Science with Creative Computation”. In: *Proceedings of the 49th ACM Technical Symposium on Computer Science Education (SIGCSE 2018)*. ACM, pp. 167–172. DOI: 10.1145/3159450.3159539.

Yang, Qian et al. (2018). “Grounding Interactive Machine Learning Tool Design in How Non-Experts Actually Build Models”. In: *Proceedings of the 2018 Designing Interactive Systems Conference*. ACM, pp. 573–584. DOI: 10.1145/3196709.3196729.

Zhou, Yicong and Daniel Schofield (2024). “Developing a Conceptual Framework for AI Literacy in Higher Education”. In: *Journal of Learning Development in Higher Education* 31. DOI: 10.47408/jldhe.vi32.1354.

Appendices

A. Data Collection Instruments

This appendix presents the data collection instruments administered during the workshop study conducted on February 12, 2026. All instruments are reproduced as administered to participants.

A.1. Workshop Recruitment Poster

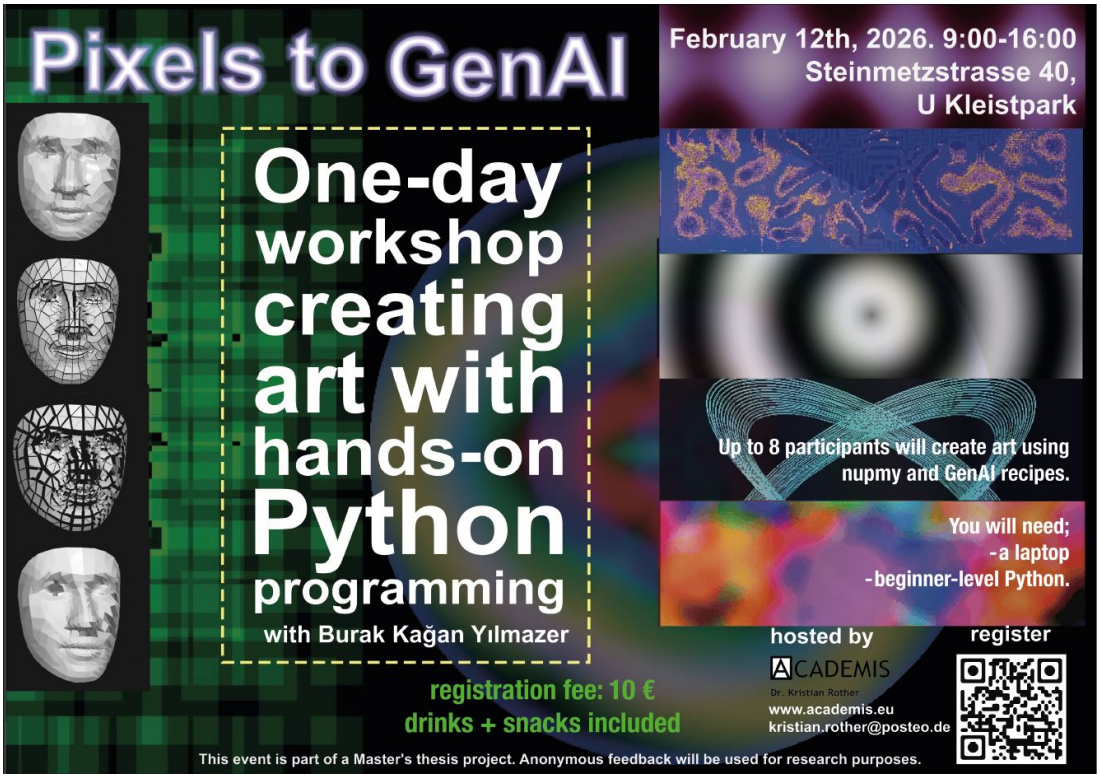


Figure A.1.: Recruitment poster distributed to advertise the workshop (February 12, 2026).

A.2. Informed Consent Form

INFORMED CONSENT FOR RESEARCH PARTICIPATION

Study Title: Bridging Computational Foundations to Generative AI: A Design-Based Framework for Progressive Creative Coding Education
Facilitator: MSc. Burak Kagan Yilmazer
Faculty Supervisor: Dr. Kristian Rother

Institution: SRH University of Applied Sciences, Germany
Program: Master of Science in Big Data & Artificial Intelligence

Purpose of the Study

You are invited to participate in a research study evaluating an educational curriculum designed to teach generative AI concepts through creative coding. The study aims to understand how learners progress from basic array manipulation to understanding neural networks and generative models.

What You Will Be Asked to Do

If you agree to participate, you will:

1. **Before the workshop (30–35 minutes, online):**
   - Complete demographic questions
   - Complete a self-efficacy survey about your confidence with programming concepts[1]
   - Complete a brief knowledge assessment
2. **During the workshop (7 hours, in-person on February 12, 2026):**
   - Work through 6 educational modules on creative coding and AI
   - Complete brief surveys (5 minutes each) after each module
   - Participate in group activities with other learners
3. **After the workshop (15 minutes, online):**
   - Complete a post-workshop knowledge assessment
   - Complete a post-workshop self-efficacy survey (see earlier footnote; not administered)
4. **Follow-up (30 minutes, 2–3 weeks later, online/video call):**
   - Participate in a brief interview about your learning experience
   - Complete a short follow-up survey

1. The self-efficacy survey (CPSES) described in the consent form was planned but ultimately not administered. The instrument was removed from the protocol before the workshop to reduce participant burden and keep the session within its time budget.

Potential Risks and Discomforts

The risks associated with this study are minimal:

- You may experience mild frustration if exercises prove challenging
- You will be spending approximately 7 hours on learning activities, which may cause fatigue
- Breaks are scheduled throughout to minimize discomfort

Potential Benefits

By participating, you may:

- Learn valuable skills in creative coding and generative AI
- Gain understanding of neural network fundamentals
- Receive a certificate of participation
- Contribute to educational research that will benefit future learners

Confidentiality

Your privacy will be protected:

- You will be assigned a participant ID code; your name will not appear in any data files
- All data will be stored on encrypted, password-protected devices
- Only the research team will have access to identifiable information
- Published results will use aggregate data only; no individual will be identifiable
- The mapping between your name and ID code will be destroyed after the study concludes

Voluntary Participation

Your participation is entirely voluntary:

- You may decline to answer any question
- You may withdraw from the study at any time without penalty
- Withdrawal will not affect any relationship you have with the researchers or institution
- You may request deletion of your data at any time before final analysis

Questions

If you have questions about this study, please contact:

- Burak Kagan Yilmazer: burak.kagan@protonmail.com
- Dr. Kristian Rother: kristian.rother@posteo.de

Consent Statement

By checking the boxes below and signing, I confirm that:

☐ I have read and understood the information above

☐ I have had the opportunity to ask questions

☐ I understand that my participation is voluntary

☐ I understand that I may withdraw at any time

☐ I consent to participate in this research study

☐ I consent to the use of anonymized data in academic publications

Participant Name: _____

Signature: _____ **Date:** _____

A.3. Participant Demographic Survey

Instructions: Please answer the following questions about yourself. This information helps us understand who is participating in our study and how to interpret the results. All responses are confidential.

Participant ID: _____ *(Assigned by researcher)*

A.3.1. Section A: Basic Information

A1. What is your age?

☐ 18–24

☐ 25–34

☐ 35–44

☐ 45–54

☐ 55 or older

☐ Prefer not to say

A2. What is your gender?

☐ Male

☐ Female

☐ Non-binary

☐ Prefer to self-describe: _____

☐ Prefer not to say

A3. What is your highest completed level of education?

☐ High school / Secondary school

☐ Bachelor's degree

☐ Master's degree

☐ Doctoral degree

☐ Other: _____

A4. What is/was your primary field of study?

☐ Computer Science / Software Engineering

☐ Data Science / Statistics

☐ Art / Design / Visual Arts

☐ Media / Communication

☐ Engineering (non-CS)

☐ Other: _____

A.3.2. Section B: Current Role

B1. What best describes your current primary role?

☐ Student (undergraduate)

☐ Student (graduate/postgraduate)

☐ Professional (technical role—developer, data scientist, etc.)

☐ Professional (creative role—designer, artist, etc.)

☐ Professional (other)

☐ Educator / Teacher

☐ Researcher

☐ Other: _____

B2. How many years of experience do you have with programming?

☐ Less than 1 year

☐ 1–2 years

☐ 3–5 years

☐ 6–10 years

☐ More than 10 years

A.3.3. Section C: Technical Background (Prior Experience Inventory)

C1. Rate your current proficiency with Python:

1 No experience

2 Beginner (can write simple scripts with guidance)

3 Intermediate (can write scripts independently)

4 Advanced (comfortable with complex programs)

5 Expert (can develop large applications)

C2. Rate your current proficiency with NumPy:

1 Never used it

2 Basic awareness (know what it is)

3 Beginner (can create and manipulate arrays)

4 Intermediate (comfortable with broadcasting, slicing)

5 Advanced (use regularly for complex operations)

C3. Have you had any prior experience with the following? (Check all that apply)

Topic	No Experience	Self-Taught	Formal Course	Profes
Image processing	☐	☐	☐	
Creative coding (Processing, p5.js, etc.)	☐	☐	☐	
Machine learning basics	☐	☐	☐	
Neural networks	☐	☐	☐	
Generative AI (GANs, Diffusion, etc.)	☐	☐	☐	
TouchDesigner	☐	☐	☐	

C4. Which of these generative AI tools have you used? (Check all that apply)

☐ DALL-E / ChatGPT image generation

☐ Midjourney

☐ Stable Diffusion

☐ RunwayML

☐ Other generative AI tools: _____

☐ None of the above

A.3.4. Section D: Motivation

D1. What is your primary motivation for participating in this workshop? (Select up to 2)

☐ Learn about generative AI for creative projects

☐ Understand how neural networks work

☐ Improve Python/NumPy skills through visual projects

☐ Professional development / career advancement

☐ Academic interest / coursework

☐ Personal curiosity

☐ Other: _____

D2. What do you hope to be able to do after completing this workshop?

_____

_____

_____

A.4. Knowledge Assessment Tests

The pre-test and post-test each contained 24 parallel-form multiple-choice items organized across six sections corresponding to the workshop modules. Both forms assessed the same conceptual domains with differently worded items to minimize test–retest memorization effects. Each item had four response options (A–D) plus an "I don't know" option to discourage guessing. Items were scored dichotomously (1 = correct, 0 = incorrect or "I don't know"). Correct answers are marked with ►.

A.4.1. Pre-Test

Section 1: RGB Basics (Items 1–4)

Q1. In NumPy, when representing a color image as a 3D array with shape (100, 200, 3), what do these dimensions represent?

A) Width, height, color channels

► B) Height, width, color channels

B) Color channels, height, width

C) Height, color channels, width

E) I don't know

Q2. In the RGB color model, what color results from Red=255, Green=255, Blue=0?

A) Cyan

B) Magenta

► C) Yellow

C) White

E) I don't know

Q3. A grayscale image in RGB format is created by:

A) Setting all three RGB channels to different values

► B) Setting all three RGB channels to the same value

B) Using only the Red channel

C) Using only the Blue channel

E) I don't know

Q4. For uint8 pixel data type, what is the valid range for each color channel?

A) 0 to 1

B) -128 to 127

► C) 0 to 255

C) 0 to 1000

E) I don't know

Section 2: Cellular Automata (Items 5–8)

Q5. In Conway's Game of Life, a dead cell becomes alive when it has exactly how many live neighbors?

A) 1

B) 2

► C) 3

C) 4

E) I don't know

Q6. Which Game of Life pattern oscillates between two states with period 2?

A) Block

► B) Blinker

B) Glider

C) Loaf

E) I don't know

Q7. In the Moore neighborhood, how many cells surround each center cell?

A) 4

B) 6

► C) 8

C) 9

E) I don't know

Q8. What happens to a live cell with fewer than 2 live neighbors?

A) It survives

► B) It dies from underpopulation

B) It reproduces

C) It moves to an adjacent cell

E) I don't know

Section 3: Convolution (Items 9–12)

Q9. In image convolution, what is the primary purpose of the kernel?

A) To resize the image

► B) To define weights for combining pixel values in a neighborhood

B) To convert the image to grayscale

C) To compress the image file size

E) I don't know

Q10. When applying a blur kernel, what is the critical requirement for kernel values?

A) They must all be negative

B) They must sum to zero

► C) They must sum to one

C) They must be prime numbers

E) I don't know

Q11. What kernel type has a large positive center value with negative surrounding values?

A) Blur kernel

B) Identity kernel

► C) Sharpen kernel

C) Edge detection kernel

E) I don't know

Q12. Edge detection kernels typically have values that sum to:

A) 1

► B) 0

B) -1

C) 9

E) I don't know

Section 4: Fractal Square (Items 13–16)

Q13. What mathematical property is fundamental to fractals?

A) Randomness

► B) Self-similarity at different scales

B) Constant color

C) Fixed size

E) I don't know

Q14. In recursive fractal algorithms, what determines when recursion stops?

A) Random chance

► B) A base case condition (e.g., depth = 0)

B) User input

C) Available memory

E) I don't know

Q15. In the fractal square algorithm dividing a region into a 3×3 grid, which section is filled at each recursion level?

A) The four corners

► B) The center square

B) The edges

C) A random section

E) I don't know

Q16. What does the "depth" parameter control in fractal generation?

A) The color intensity

► B) The level of detail/number of recursion levels

B) The image resolution

C) The speed of rendering

E) I don't know

Section 5: Perceptron (Items 17–20)

Q17. A perceptron is a type of:

A) Unsupervised learning algorithm

► B) Binary classifier

B) Clustering algorithm

C) Regression model

E) I don't know

Q18. In a perceptron, the step activation function outputs 1 when:

A) The weighted sum is less than zero

► B) The weighted sum equals zero or is greater than zero

B) The weighted sum is exactly one

C) The bias is positive

E) I don't know

Q19. According to Minsky and Papert (1969), what is a fundamental limitation of a single perceptron?

A) It cannot learn from data

B) It cannot solve linearly separable problems

► C) It cannot solve the XOR problem

C) It requires too much memory

E) I don't know

Q20. In the perceptron learning rule, when does weight update occur?

A) After every input, regardless of correctness

► B) Only when the prediction is incorrect

B) Only when the prediction is correct

C) At random intervals

E) I don't know

Section 6: DDPM Basics (Items 21–24)

Q21. In diffusion models, what is the forward diffusion process?

A) Generating images from noise

► B) Gradually adding noise to a clean image

B) Training the neural network

C) Compressing the image

E) I don't know

Q22. What does the neural network in a DDPM learn to predict?

A) The original clean image directly

► B) The noise that was added at each timestep

B) The final output image

C) The optimal learning rate

E) I don't know

Q23. Compared to GANs, what is a key advantage of diffusion models during training?

A) Faster training time

B) Single-pass image generation

► C) More stable training with simple MSE loss

C) Smaller model size

E) I don't know

Q24. What architecture is commonly used for the noise prediction network in DDPMs?

A) ResNet

B) VGG

► C) U-Net

C) AlexNet

E) I don't know

A.4.2. Post-Test

Section 1: RGB Basics (Items 1–4)

Q1. When accessing a specific pixel in a NumPy image array using `array[y, x]`, why does the y-coordinate come before x?

A) It's an arbitrary convention with no reason

B) It matches how monitors display pixels

C) It makes the code run faster

► D) NumPy uses row-major order where rows (height/y) come first

E) I don't know

Q2. What color results from combining Green=255 and Blue=255 with Red=0?

A) Yellow

B) Magenta

► C) Cyan

C) White

E) I don't know

Q3. How many distinct colors can be represented using 24-bit true color (8 bits per RGB channel)?

A) 256 colors

B) 65,536 colors

► C) Approximately 16.7 million colors

C) Unlimited colors

E) I don't know

Q4. The RGB color model is called "additive" because:

A) You subtract colors to create new ones

► B) Combining maximum values of all channels produces white (light adds together)

B) Colors are added to a database

C) The file size increases with more colors

E) I don't know

Section 2: Cellular Automata (Items 5–8)

Q5. In Conway's Game of Life, a live cell survives to the next generation when it has:

A) 1 or 2 neighbors

► B) 2 or 3 neighbors

B) 3 or 4 neighbors

C) Any number of neighbors

E) I don't know

Q6. Which pattern type in Conway's Game of Life moves across the grid over time?

A) Still life

B) Oscillator

► C) Spaceship

C) Garden of Eden

E) I don't know

Q7. A "block" pattern (2×2 square of live cells) in Conway's Game of Life is classified as:

► A) A still life

A) An oscillator

B) A spaceship

C) A methuselah

E) I don't know

Q8. When a live cell has more than 3 neighbors in Conway's Game of Life, what occurs?

A) The cell survives

B) The cell splits into two

► C) The cell dies from overpopulation

C) Nothing happens

E) I don't know

Section 3: Convolution (Items 9–12)

Q9. During convolution, how is the output value at each position calculated?

A) By finding the maximum pixel value in the neighborhood

► B) By computing the weighted sum of the neighborhood multiplied by the kernel

B) By averaging all pixels in the image

C) By sorting the pixel values

E) I don't know

Q10. An identity kernel (no change to image) has what characteristic?

► A) A single 1 in the center with 0s elsewhere

A) All values are 1

B) All values are negative

C) Values sum to zero

E) I don't know

Q11. Why do edge detection kernels respond strongly to areas where pixel values change rapidly?

A) Because they have large positive values

► B) Because positive and negative values cancel in uniform regions but not at edges

B) Because they blur the image first

C) Because they only look at corner pixels

E) I don't know

Q12. When applying a 3×3 averaging (blur) kernel, what value should each cell contain for proper normalization?

A) 1

B) 0.5

► C) 1/9

C) 3

E) I don't know

Section 4: Fractal Square (Items 13–16)

Q13. Fractals exhibit the same patterns at different scales. This property is called:

A) Recursion

► B) Self-similarity

B) Iteration

C) Transformation

E) I don't know

Q14. In a recursive fractal algorithm, what happens if there is no base case?

A) The fractal becomes more detailed

B) The colors change

► C) The recursion continues infinitely (stack overflow)

C) The image becomes smaller

E) I don't know

Q15. In the fractal square pattern, when the region is divided into a 3×3 grid, recursion occurs on which parts?

A) Only the center square

B) All nine squares

► C) The four corner squares

C) The four edge squares

E) I don't know

Q16. Increasing the recursion depth in fractal generation results in:

A) Larger overall image size

► B) More levels of nested detail

B) Fewer colors

C) Faster rendering time

E) I don't know

Section 5: Perceptron (Items 17–20)

Q17. The perceptron computes a weighted sum of inputs plus a bias, then applies:

A) A logarithmic function

► B) A step (threshold) function

B) A polynomial function

C) A random function

E) I don't know

Q18. What does the bias term allow the perceptron to do?

► A) Shift the decision boundary away from the origin

A) Learn faster

B) Handle more inputs

C) Reduce memory usage

E) I don't know

Q19. For linearly separable data, the perceptron learning algorithm is guaranteed to:

A) Never converge

► B) Converge to a solution that correctly classifies all training points

B) Find the optimal solution

C) Produce random results

E) I don't know

Q20. In the perceptron learning rule, what determines the direction of weight adjustment?

A) The learning rate only

B) Random chance

► C) The error (difference between target and prediction) multiplied by the input

C) The number of training iterations

E) I don't know

Section 6: DDPM Basics (Items 21–24)

Q21. During DDPM sampling (image generation), what is the starting point?

A) A clean image from the dataset

► B) Pure Gaussian noise

B) A low-resolution image

C) A template image

E) I don't know

Q22. The training loss function in DDPM is:

► A) Mean squared error between predicted noise and actual noise

A) Adversarial loss between generator and discriminator

B) Cross-entropy loss

C) Perceptual loss

E) I don't know

Q23. DDIM (Denoising Diffusion Implicit Models) enables what improvement over standard DDPM?

A) Better image quality

► B) Faster sampling with fewer steps

B) Smaller model size

C) Reduced training time

E) I don't know

Q24. Why does the U-Net in DDPM include timestep embeddings?

A) To track training progress

B) To save computation time

► C) To inform the network which noise level it should predict for

C) To increase image resolution

E) I don't know

A.5. NASA Task Load Index (NASA-TLX)

The Raw NASA Task Load Index (Hart et al. 1988) was administered after each module to assess perceived cognitive load. The raw TLX format was used (no weighting procedure). Participants rated each subscale on a 21-point scale ranging from 0 to 100 in 5-point increments.

Subscale	Prompt	Scale Anchors
Mental Demand	How mentally demanding was the task?	Very Low – Very High
Physical Demand	How physically demanding was the task?	Very Low – Very High
Temporal Demand	How hurried or rushed was the pace of the task?	Very Low – Very High
Performance	How successful were you in accomplishing what you were asked to do?	Perfect – Failure
Effort	How hard did you have to work to accomplish your level of performance?	Very Low – Very High
Frustration	How insecure, discouraged, irritated, stressed, and annoyed were you?	Very Low – Very High

Each subscale was presented with a continuous line marked with 21 gradations. Participants placed a mark on the line corresponding to their perceived level for each dimension.

A.6. Exit Ticket Questionnaire

The exit ticket was administered after each module to capture immediate reflections on learning. One exit ticket was completed per module (six total per participant).

EXIT TICKET

Participant ID: _____ **Module:** _____

Q1. What was the MOST important concept you learned in this module?

_____

_____

_____

Q2. How does this module connect to what you learned earlier? *(Specifically name any previous module concepts you used or built upon)*

_____

_____

_____

Q3. What was the MOST confusing or challenging part?

Q4. Rate your understanding of this module's main concepts:

Did not understand		Somewhat understood		Fully understood
1	2	3	4	5
☐	☐	☐	☐	☐

Q5. One thing you want to explore more:

A.7. Facilitator Observation Protocol

The facilitator observation protocol was used to record behavioral data during each module session. One protocol was completed per module by the facilitator/observer.

FACILITATOR OBSERVATION PROTOCOL

Observer: _____ **Module:** _____

Start Time: _____ **End Time:** _____ **Participants Present:** _____

A.7.1. Section A: Behavioral Indicators

Tally marks every 5 minutes.

Time	Engaged (coding)	Off-task	Seeking Help	Helping Others	Technical Issue
0–5 min					
5–10 min					
10–15 min					
15–20 min					
20–25 min					
25–30 min					
30–35 min					
35–40 min					
40–45 min					
45–50 min					

Definitions:

- **Engaged:** Actively typing, reading module, viewing output
- **Off-task:** Phone use, unrelated browsing, extended unfocused periods
- **Seeking Help:** Raising hand, calling facilitator, asking neighbor
- **Helping Others:** Explaining to neighbor, pointing at screen
- **Technical Issue:** Software error, website not loading, etc.

A.7.2. Section B: Help Requests Log

Time	Participant ID	Nature of Request	Resolution

Nature of Request Categories:

- **TC** = Technical (setup, errors)
- **CC** = Conceptual (understanding concept)
- **PC** = Procedural (what to do next)
- **DB** = Debugging (code not working)

A.7.3. Section C: Notable Observations

Breakthrough Moments (participant shows clear understanding):

Struggle Points (multiple participants stuck on same thing):

Unexpected Events:

Quotes Worth Noting (verbatim participant comments):

A.7.4. Section D: Module Completion Summary

Participant	Completed Main Exercise		Completed Challenge		Notes
P01	☐ Yes	☐ No	☐ Yes	☐ No	
P02	☐ Yes	☐ No	☐ Yes	☐ No	
P03	☐ Yes	☐ No	☐ Yes	☐ No	
P04	☐ Yes	☐ No	☐ Yes	☐ No	
P05	☐ Yes	☐ No	☐ Yes	☐ No	
P06	☐ Yes	☐ No	☐ Yes	☐ No	
P07	☐ Yes	☐ No	☐ Yes	☐ No	
P08	☐ Yes	☐ No	☐ Yes	☐ No	

A.7.5. Section E: Overall Module Assessment

Pacing: ☐ Too fast ☐ About right ☐ Too slow

Engagement Level: ☐ Low ☐ Moderate ☐ High

Difficulty Level (observed): ☐ Too easy ☐ Appropriate ☐ Too challenging

Additional Notes:

B. Supplementary Data Tables

This appendix presents the complete data collected during the workshop study. All tables report raw scores as recorded; summary statistics derived from these data appear in Chapter 5.

B.1. Full Knowledge Test Results

Table B.1.: Pre-test and post-test scores by participant and section. Scores represent correct items per section (max 6 per section, 24 total). IDK = "I don't know" responses.

PID	Sec A		Sec B		Sec C		Sec D		Total	
	Score	IDK	Score	IDK	Score	IDK	Score	IDK	Score	IDK
Pre-Test ($n = 9$)										
P01	0	4	1	4	1	4	3	0	5	12
P02	3	3	0	6	2	4	0	5	5	18
P03	2	3	4	2	3	2	0	6	9	13
P04	0	6	0	6	0	6	0	6	0	24
P05	1	4	0	6	0	6	0	6	1	22
P06	1	5	0	6	1	5	0	6	2	22
P07	1	2	0	6	0	5	0	6	1	19
P08	0	4	0	2	1	2	1	4	2	12
P09	2	3	2	2	0	6	0	6	4	17
Post-Test ($n = 8$; P04 did not complete post-test)										
P01	5	0	3	0	5	1	5	0	18	1
P02	6	0	6	0	6	0	6	0	24	0
P03	5	0	3	0	5	1	1	4	14	5
P05	2	0	2	1	3	0	1	3	8	4
P06	2	1	3	1	1	3	0	6	6	11
P07	3	1	3	1	4	0	1	4	11	6
P08	3	2	1	2	1	2	1	4	6	10
P09	3	1	4	0	3	2	0	6	10	9

B.2. NASA-TLX Scores by Participant and Module

Table B.2.: NASA-TLX raw scores by participant and module (scale 1–20). Performance is reverse-scored (higher = greater workload). Overall workload is the unweighted mean of all six subscales.

PID	Mental	Physical	Temporal	Perf.[a]	Effort	Frust.	Overall
M1: RGB Basics (HOD)							
P01	2	1	3	15	14	11	7.67
P02	7	2	11	18	11	3	8.67
P03	2	1	2	20	3	1	4.83
P04	9	1	10	11	8	6	7.50
P05	13	7	5	12	17	4	9.67
P06	5	1	10	17	8	7	8.00
P07	10	1	7	20	7	10	9.17
P08	1	1	1	20	1	1	4.17
P09	8	4	16	17	8	7	10.00
M2: Cellular Automata (HOD)							
P01	16	1	16	12	12	9	11.00
P02	13	2	13	19	13	2	10.33
P03	9	1	14	12	6	1	7.17
P04	18	1	18	17	18	15	14.50
P05	20	7	16	14	20	14	15.17
P06	11	3	7	6	8	4	6.50
P07	20	8	15	20	11	9	13.83
P08	1	1	1	20	11	1	5.83
P09	13	11	17	16	15	13	14.17
M3: Convolution (CDD)							
P01	5	1	11	16	6	1	6.67
P02	14	3	10	19	9	2	9.50
P03	5	1	7	20	5	1	6.50
P04	9	5	13	17	13	13	11.67
P05	20	5	6	14	19	10	12.33
P06	15	8	15	6	13	14	11.83
P07	20	11	18	20	13	9	15.17
P08	1	1	1	20	5	1	4.83
P09	8	8	8	19	9	13	10.83
M4: Fractal Square (HOD)							
P01	7	1	3	20	13	1	7.50
P02	16	3	10	17	11	3	10.00
P03	8	1	8	15	10	1	7.17
P04	8	1	10	11	7	12	8.17
P05	18	7	8	19	16	9	12.83
P06	7	5	5	19	12	5	8.83
P07	10	10	4	18	6	5	8.83

Table B.2.: (continued)

PID	Mental	Physical	Temporal	Perf.[a]	Effort	Frust.	Overall
P08	1	1	1	15	2	1	3.50
P09	10	7	10	19	7	10	10.50
M5: Perceptron (CDD)							
P01	8	1	4	17	6	1	6.17
P02	15	3	12	18	14	3	10.83
P03	14	1	7	13	13	6	9.00
P04	15	13	11	15	19	14	14.50
P05	20	6	7	13	20	12	13.00
P06	16	7	13	12	14	15	12.83
P07	20	11	16	4	11	15	12.83
P08	1	1	1	13	5	1	3.67
P09	17	11	14	13	13	13	13.50
M6: DDPM Basics (CDD)							
P01	8	1	14	20	11	4	9.67
P02	19	6	15	11	14	14	13.17
P03	9	1	17	6	3	7	7.17
P04	16	1	15	14	18	18	13.67
P05	20	7	12	4	17	13	12.17
P06	15	5	15	10	14	12	11.83
P07	17	11	17	17	11	9	13.67
P08	1	1	1	17	7	1	4.67
P09	16	8	13	5	11	7	10.00

[a] Performance scores are inverted (21 − raw score) so that higher values indicate greater workload, consistent with the other five subscales.

B.3. Exit Ticket Coded Responses

The following tables present the qualitative coding of exit ticket open-ended responses. Responses were coded using the codebook presented in Table B.3. Multiple codes were assigned where applicable, separated by the pipe character (|). Empty cells indicate no response was provided. Participant responses are reproduced verbatim, including original spelling and grammar.

Table B.3.: Qualitative codebook used for exit ticket analysis. Thirty-one codes across five categories were applied to open-ended responses from Questions 1, 2, 3, and 5.

Code	Category	Definition	Example Quote
Learning			

Table B.3.: (continued)

Code	Category	Definition	Example Quote
LEARN_PROCEDURAL	Learning	Participant learned a specific technique, operation, or how-to	"How to create a gradiant pattern" (P1-M1)
LEARN_CONCEPTUAL	Learning	Participant understood a core concept or principle	"The Numpy array representation of an image" (P2-M1)
LEARN_APPLIED	Learning	Participant connected concept to real-world or creative applications	"Kernel values gave me a lot of insights how creative apps could be working behind the scenes" (P7-M3)
LEARN_TOOL_SETUP	Learning	Primary takeaway was about tools or environment setup	"Installing and running Python in VS Code" (P6-M1)
LEARN_META	Learning	Participant reflects on own learning state rather than specific content	"I didn't yet know a lot about neural networks" (P3-M5)
Connections			
CONN_EXPLICIT	Connections	Names a specific prior module, concept, or technique	"The kernel was introduced in module 1.2.2" (P2-M3)
CONN_VAGUE	Connections	Affirms connection exists but does not specify which concept or how	"I think it connects well" (P1-M3)
CONN_NONE	Connections	Explicitly states no connection seen	"Don't see any connections yet" (P8-M2)
CONN_EXTERNAL	Connections	References prior knowledge from outside the curriculum	"I am familiar with p5.js variables" (P7-M2)
CONN_GAP	Connections	Notes that skipped modules create disconnect	"We jumped a lot of modules, no direct connection" (P2-M6)
CONN_UNCERTAIN	Connections	Expresses uncertainty about whether or how modules connect	"Not sure how it connects to 1.1.1 exactly" (P5-M2)
Challenges			
CHAL_MATH	Challenges	Difficulty with mathematical notation, formulas, or numerical reasoning	"the math functions are very unfamiliar to me" (P5-M5)
CHAL_CODE	Challenges	Difficulty reading, writing, or understanding Python code	"I think I still struggle with reading code" (P7-M3)
CHAL_TECH_SETUP	Challenges	Issues with installation, dependencies, or IDE configuration	"Dependency issues, RGB values in Python" (P6-M1)
CHAL_TIME_PACE	Challenges	Insufficient time to complete exercises or content felt too dense	"I think it wasn't enough time to soak everything in as a beginner" (P7-M2)
CHAL_CONCEPTUAL	Challenges	Difficulty grasping abstract ideas or forming mental models	"Figuring out how change in kernel values produces change in image" (P5-M3)
CHAL_OVERLOAD	Challenges	Explicit mention of too much content or complexity	"To much reading, complex content" (P9-M5)
CHAL_NAVIGATION	Challenges	Difficulty navigating documentation or finding output files	"It was challaning to understand the expected outcome and to learn how to navigate the files" (P6-M2)

Table B.3.: (continued)

Code	Category	Definition	Example Quote
CHAL_NONE	Challenges	Participant explicitly reports no difficulty	"I think everything was okay" (P8-M1)
CHAL_VISUAL_GAP	Challenges	Missing step-by-step visualization or intermediate output	"There was no display of the intermediate grid states" (P2-M2)
CHAL_COPY_PASTE	Challenges	Exercises involved copying code without genuine understanding	"The exercises are copy-paste without undertanding" (P9-M5)
Engagement			
ENG_DEPTH	Engagement	Wants to go deeper into the current topic	"Deeper understanding of recursion" (P9-M4)
ENG_CREATIVE	Engagement	Interested in creative applications or experimenting with parameters	"Using different geometric shapes and sub-division patterns" (P2-M4)
ENG_CONTINUE	Engagement	Wants to finish exercises or have more time with material	"run out of time before I could do the bonus exercise" (P2-M1)
ENG_PERSEVERE	Engagement	Determination to keep going despite difficulty	"I just want to keep exposing myself to these concepts until I get there" (P7-M5)
ENG_BREADTH	Engagement	Wants to explore related but different topics	"I would like to learn more about image processing" (P1-M3)
ENG_SEQUENTIAL	Engagement	Explicitly wants to go through curriculum in order	"Going through all the chapters in order" (P2-M6)
Experience			
EXP_BEGINNER	Experience	Self-identifies as beginner; cites it as context for difficulty	"Understanding how to read python code (I am beginner)" (P5-M1)
EXP_NOVELTY_POS	Experience	Novelty experienced positively	"I never knew abot any of this, so in geenral it was really interesting" (P7-M5)
EXP_HANDS_ON	Experience	References playing, experimenting, or modifying code as learning method	"It was more intuitive to learn the match concept by playing around with the numbers in the code" (P5-M4)

Table B.4.: Coded exit ticket responses—Q1: "What was the MOST important concept you learned in this module?"

PID	Module	Response	Codes
P01	M1	How to create a gradiant pattern	LEARN_PROCEDURAL
P02	M1	The Numpy array representation of an image	LEARN_CONCEPTUAL
P03	M1	How Numpy image arrays can be filled	LEARN_CONCEPTUAL | LEARN_PROCEDURAL
P04	M1	How RGB works	LEARN_CONCEPTUAL

Table B.4.: (continued)

PID	Module	Response	Codes
P05	M1	The basics of reading python code and how to create an image with colours and greyscale	LEARN_PROCEDURAL | EXP_BEGINNER
P06	M1	Installing and running Python in VS Code	LEARN_TOOL_SETUP
P07	M1	The basics of RGB and getting a better grasp of what is what	LEARN_CONCEPTUAL
P08	M1	Never used PIL library	LEARN_TOOL_SETUP | EXP_NOVELTY_POS
P09	M1	Array slicing and the diffrence between hight/width, when the width and height are reversed in order. Also channales and greyscale	LEARN_CONCEPTUAL
P01	M2	Computational beauty of nature and cellular automata	LEARN_CONCEPTUAL
P02	M2	Learning about cellular automata / game of life	LEARN_CONCEPTUAL
P03	M2	Conways game of life basics	LEARN_CONCEPTUAL
P04	M2	How many cells are needed for the cell to be alive	LEARN_CONCEPTUAL
P05	M2	How the game of life works as a concept	LEARN_CONCEPTUAL
P06	M2	Creating a grid of cells	LEARN_PROCEDURAL
P07	M2	The concept of neighbors was interesting	LEARN_CONCEPTUAL | EXP_NOVELTY_POS
P08	M2	How to use the cell automata	LEARN_PROCEDURAL
P09	M2	The game of life rules and how arrays interact to create the game of life	LEARN_CONCEPTUAL | LEARN_PROCEDURAL
P01	M3	To learn about convoluiton	LEARN_CONCEPTUAL
P02	M3	Learning about convolution kernels, and how they can be used to apply different affects on images	LEARN_CONCEPTUAL | LEARN_APPLIED
P03	M3	Image processing using kernels/filters	LEARN_CONCEPTUAL | LEARN_PROCEDURAL
P04	M3	Some basic image processing	LEARN_CONCEPTUAL
P05	M3	Kernel and how by changing its values we can influence the output of the image	LEARN_CONCEPTUAL
P06	M3	Pixel manipulation	LEARN_PROCEDURAL
P07	M3	Kernel values gave me a lot of insights how creative apps could be working behind the scenes	LEARN_APPLIED
P08	M3	Doing convolution and using kernel	LEARN_PROCEDURAL
P09	M3		
P01	M4	I learned about fractals	LEARN_CONCEPTUAL
P02	M4	Self similarity of fractals, and that these shapes exist a lot in nature	LEARN_CONCEPTUAL | LEARN_APPLIED
P03	M4	Fractals	LEARN_CONCEPTUAL
P04	M4	Recursion	LEARN_CONCEPTUAL
P05	M4	How fractals work and the concept behind recursion	LEARN_CONCEPTUAL
P06	M4	Fractals and recursion	LEARN_CONCEPTUAL
P07	M4	I understand how recursion works better now	LEARN_CONCEPTUAL

Table B.4.: (continued)

PID	Module	Response	Codes
P08	M4	Create a fractal square	LEARN_PROCEDURAL
P09	M4	That I can recursivley divide a square to create a fractal	LEARN_CONCEPTUAL | LEARN_PROCEDURAL
P01	M5	The concept of Perceptron was interesting	LEARN_CONCEPTUAL
P02	M5	What a single perceptron does and how	LEARN_CONCEPTUAL
P03	M5	I didn't yet know a lot about neural networks	LEARN_META
P05	M5	The concepts and fundamentals behind perceptron	LEARN_CONCEPTUAL
P06	M5		
P07	M5	I never knew abot any of this, so in geenral it was really interesting	EXP_NOVELTY_POS
P08	M5	What is perceptron	LEARN_CONCEPTUAL
P09	M5	The concept behind the math, for example: weights etc.	LEARN_CONCEPTUAL
P01	M6	Denoising diffusion models	LEARN_CONCEPTUAL
P02	M6	The forward and reverse diffusion process	LEARN_CONCEPTUAL
P03	M6	How an image is noised and denoised through multiple steps, and thus using this to train a model to generate images	LEARN_CONCEPTUAL | LEARN_APPLIED
P05	M6	The concept on DDPM and its components	LEARN_CONCEPTUAL
P07	M6	The concept of forward and reverse process	LEARN_CONCEPTUAL
P08	M6		
P09	M6	General idea of diffusion	LEARN_CONCEPTUAL

Table B.5.: Coded exit ticket responses—Q2: "How does this module connect to what you learned earlier?"

PID	Module	Response	Codes
Note: M1 (RGB Basics) was the first module; Q2 was not applicable.			
P01	M2	It connects the concepts with arrays, colors and slicing. Maybe I would use some other colors instead of just black to bring the color concepts from the first module to second	CONN_EXPLICIT | ENG_CREATIVE
P02	M2	The array concept for image processing from module 1.1.1	CONN_EXPLICIT
P03	M2	np.arrays	CONN_EXPLICIT
P05	M2	Not sure how it connects to 1.1.1 exactly. But understanding how pixels can be presented in greyscale helped	CONN_UNCERTAIN | CONN_EXPLICIT
P06	M2	Learning how grids work and how to define rows etc	CONN_VAGUE

Table B.5.: (continued)

PID	Module	Response	Codes
P07	M2	I am familiar with p5.js variables. Comparing the two concepts made me understanding the module a bit better	CONN_EXTERNAL
P08	M2	Don't see any connections yet	CONN_NONE
P09	M2	array slicing and grid[row:height]	CONN_EXPLICIT
P01	M3	I think it connects well	CONN_VAGUE
P02	M3	The kernel was introduced in module 1.2.2	CONN_EXPLICIT
P03	M3	Concept of kernels	CONN_EXPLICIT
P05	M3	Previous module's concept of game of life is useful here to understand how the kernels can be manipulated. Still not entierly certain how the changes happen exactly though	CONN_EXPLICIT | CONN_UNCERTAIN
P06	M3	Learning how to manipulate image in various ways	CONN_VAGUE
P07	M3	nothing similar	CONN_NONE
P08	M3	Don't see any connection	CONN_NONE
P09	M3	row:col	CONN_EXPLICIT
P01	M4	I think it connects well	CONN_VAGUE
P02	M4	Using a 3x3 grid, RGB color channels	CONN_EXPLICIT
P03	M4	np,arrays	CONN_EXPLICIT
P05	M4	Not really sure how exactly fractls relate to 3.4 convolution, perhaps the way kernels and corners are related	CONN_UNCERTAIN
P06	M4	Colour theory and exporting an image	CONN_EXPLICIT
P07	M4	I havent recalled any, yet	CONN_NONE
P08	M4	No connections	CONN_NONE
P09	M4		
P01	M5	It connects well	CONN_VAGUE
P02	M5	2d arrays	CONN_EXPLICIT
P03	M5	np.arrays	CONN_EXPLICIT
P05	M5	Not sure, but I learned that this connects to all generative coding. So it is more like the fundamentals starting from scratch	CONN_UNCERTAIN | CONN_VAGUE
P06	M5		
P07	M5	couldn't connect to any concepts yet	CONN_NONE
P08	M5		
P09	M5	–	CONN_NONE
P01	M6		
P02	M6	We jumped a lot of modules, no direct connection	CONN_GAP | CONN_NONE
P03	M6	Neural networks	CONN_EXPLICIT
P05	M6	Connects to module 9, in terms of ML and giving neural networks a lot more complex task	CONN_EXPLICIT
P07	M6		

Table B.5.: (continued)

PID	Module	Response	Codes
P08	M6		
P09	M6		

Table B.6.: Coded exit ticket responses—Q3: “What was the MOST confusing or challenging part?”

PID	Module	Response	Codes
P01	M1	To create a yellow-cyan gradient	CHAL_CONCEPTUAL
P02	M1	Not really confusing. I have never worked with 3d and 2d color arrays, and I was successful.	CHAL_NONE | EXP_NOVELTY_POS
P03	M1	Remembering and figuring out how additive color mixing works (still not that confusing)	CHAL_CONCEPTUAL
P04	M1	To find out how to split my pixel in more than 1 color	CHAL_CONCEPTUAL
P05	M1	Understanding how to read python code (I am beginner)	CHAL_CODE | EXP_BEGINNER
P06	M1	Dependency issues, RGB values in Python	CHAL_TECH_SETUP
P07	M1	uint8 and datadype in general	CHAL_CONCEPTUAL
P08	M1	I think everything was okay	CHAL_NONE
P09	M1	Gradient pattern in the for loop. Scaling pixels to 0-255	CHAL_MATH | CHAL_CODE
P01	M2	Maybe in the first exercise, you could add a piece of code that could visualise each step of the transition because I feel like we expected it	CHAL_VISUAL_GAP
P02	M2	There was no display of the intermediate grid states in the output of the exercise	CHAL_VISUAL_GAP
P03	M2	VS code beign dumb by using the wrong interpreter	CHAL_TECH_SETUP
P05	M2	Actually, I was just getting the code to work and find the generated image file. Game of life basics concepts are explained well but information is very condenced in this module	CHAL_NAVIGATION | CHAL_TIME_PACE
P06	M2	It was challaning to understand the expected outcome and to learn how to navigate the files in docs	CHAL_NAVIGATION
P07	M2	I think it wasn’t enough time to soak everything in as a beginner	CHAL_TIME_PACE | EXP_BEGINNER
P08	M2	To install ImageIO and to make VS Code to work	CHAL_TECH_SETUP
P09	M2	I thought you only look at the alive cells and check how many neighbours it has. But you actually have to check every cell (dead or alive) in the grid. Also, I still don’t understand what is np.repeat and np.stack	CHAL_CONCEPTUAL | CHAL_CODE

Table B.6.: (continued)

PID	Module	Response	Codes
P01	M3	Maybe wrting the loop if you are a beginner	CHAL_CODE | EXP_BEGINNER
P02	M3	The output file of exercise 1 landed in an unexpected folder	CHAL_NAVIGATION
P03	M3	Task 3 in 'Make it your own'	CHAL_CONCEPTUAL
P05	M3	Figuring out how change in kernel values produces change in image. Translating numbers to math to visualise an image still feels confusing	CHAL_CONCEPTUAL | CHAL_MATH
P06	M3	Creating the loop	CHAL_CODE
P07	M3	I think I still struggle with reading code	CHAL_CODE | EXP_BEGINNER
P08	M3	Dont have any specific part which was challaning, but I resolved problem from previous bash to install ImageIO and make VS Code do what I want	CHAL_NONE | CHAL_TECH_SETUP
P09	M3		
P01	M4	I think that the module was very structured	CHAL_NONE
P02	M4	The first picture illustrates the 4 corners confusing	CHAL_CONCEPTUAL
P03	M4	math	CHAL_MATH
P05	M4	Understanding the math begind fracals. It was more intuitive to learn the match concept by playing around with the numbers in the code	CHAL_MATH | EXP_HANDS_ON
P06	M4	Understanding how the parameters impacted the fractals	CHAL_CONCEPTUAL
P07	M4		
P08	M4	To change corners and colours	CHAL_CONCEPTUAL
P09	M4	I still don't understand how to make different patterns	CHAL_CONCEPTUAL
P01	M5	For me was ok	CHAL_NONE
P02	M5	Why does the input go into calculating the new weights	CHAL_CONCEPTUAL
P03	M5	missing prior knowledge, I am not sure if I understand what I have done	CHAL_CONCEPTUAL | CHAL_OVERLOAD | EXP_BEGINNER
P05	M5	the math functions are very unfamiliar to me (computational math is confusing)	CHAL_MATH | EXP_BEGINNER
P06	M5		
P07	M5	everything :(	CHAL_OVERLOAD
P08	M5	Not really anything	CHAL_NONE
P09	M5	To much reading, complex content. Unfortunatelly, I feel like I din't learn much because there is too much to learn. The exercises are copy-paste without undertanding	CHAL_OVERLOAD | CHAL_TIME_PACE | CHAL_COPY_PASTE
P01	M6	I think that for me was fine	CHAL_NONE

Table B.6.: (continued)

PID	Module	Response	Codes
P02	M6	The ot files could be put into more organized folders	CHAL_NAVIGATION
P03	M6	PC slow	CHAL_TECH_SETUP
P05	M6	Actually to get the codes running was a bit complicated, but it was good to learn problem solving for making sure all components are installed and connected	CHAL_TECH_SETUP | EXP_HANDS_ON
P07	M6	Setting up	CHAL_TECH_SETUP
P08	M6		
P09	M6	Unsuccesfull PyTorch installation.	CHAL_TECH_SETUP

Table B.7.: Coded exit ticket responses—Q5: “One thing you want to explore more.”

PID	Module	Response	Codes
P01	M1		
P02	M1	run out of time before I could do the bonus exercise	ENG_CONTINUE
P03	M1	Gradients	ENG_DEPTH
P04	M1		
P05	M1	I want to continue exploring the module	ENG_CONTINUE
P06	M1	I would like more time to catch up	ENG_CONTINUE | EXP_BEGINNER
P07	M1	get more familiar with the UI of this tool and the fundamental concepts	ENG_CONTINUE | EXP_BEGINNER
P08	M1		
P09	M1	Diagonal gradient	ENG_DEPTH | ENG_CREATIVE
P01	M2	The computational beaut of the nature	ENG_BREADTH
P02	M2	Visualisation of grid state	ENG_DEPTH
P03	M2	other cellulat automatos	ENG_BREADTH
P05	M2	Continue to explore the concepts in the same module (all of it)	ENG_CONTINUE
P06	M2	I didn’t finish exercise 3, I would like to finish	ENG_CONTINUE
P07	M2	Spend more time to get myself familiar with everything	ENG_CONTINUE | EXP_BEGINNER
P08	M2		
P09	M2		
P01	M3	I would like to learn more about image processing	ENG_BREADTH
P02	M3	I am happy with what was explored	—
P03	M3	More filters	ENG_DEPTH | ENG_CREATIVE
P05	M3	How the value change influcences the image output	ENG_DEPTH

Table B.7.: (continued)

PID	Module	Response	Codes
P06	M3	Finishing the 3rd assignment	ENG_CONTINUE
P07	M3	I just want to keep going and see if I learn anything	ENG_PERSEVERE
P08	M3		
P09	M3		
P01	M4	I want to understand better the math of drawings	ENG_DEPTH
P02	M4	Using different geometric shapes and subdivision patterns	ENG_CREATIVE
P03	M4	fractal patterns, obviously	ENG_DEPTH
P05	M4	Mandelbroth, depth and recursion call	ENG_BREADTH | ENG_DEPTH
P06	M4	Changing the parameters	ENG_DEPTH | EXP_HANDS_ON
P07	M4	Play around with the given files	ENG_CONTINUE | EXP_HANDS_ON
P08	M4	To make it kind of complex with the patterns	ENG_CREATIVE
P09	M4	Deeper understanding of recursion	ENG_DEPTH
P01	M5		
P02	M5	connectng multiple perceptron	ENG_BREADTH
P03	M5		
P05	M5	How the perceptron learns depending on the change in weights	ENG_DEPTH
P06	M5		
P07	M5	I just want to keep exposing myself to these concepts until I get there	ENG_PERSEVERE
P08	M5	cover fully the topic	ENG_DEPTH
P09	M5		
P01	M6	creative coding	ENG_BREADTH
P02	M6	Going through all the chapters in order	ENG_SEQUENTIAL
P03	M6		
P05	M6	I would like to continue exploring the concepts in this module	ENG_CONTINUE
P07	M6	Hoping I will recall these experiences in future, once I dive deeper into these topics	ENG_PERSEVERE
P08	M6	Whole topic	ENG_DEPTH
P09	M6	Deeper understanding of denoising	ENG_DEPTH

B.4. Item Analysis Details

Table B.8.: Item difficulty and discrimination indices for the knowledge assessment. Difficulty = proportion correct. r_{it} = corrected item-total correlation (point-biserial). Items with zero variance on the pre-test (difficulty = .000) have no computable correlation.

Item	Section	Difficulty		Δ	Post-test	
		Pre	Post		r_{it}	Flag
Section A: Arrays & Images						
Q1	A	.111	.875	+.764	.016	LOW
Q2	A	.111	.500	+.389	.750	
Q3	A	.667	.375	−.292	.838	
Q4	A	.222	.875	+.653	.016	LOW
Q5	A	.000	.500	+.500	.021	LOW
Q6	A	.000	.500	+.500	.750	
Section B: Computational Concepts						
Q7	B	.000	.500	+.500	.606	
Q8	B	.111	.500	+.389	.021	LOW
Q9	B	.222	.750	+.528	.187	LOW
Q10	B	.222	.375	+.153	.122	LOW
Q11	B	.111	.625	+.514	.675	
Q12	B	.111	.375	+.264	.167	LOW
Section C: Neural Networks						
Q13	C	.333	.500	+.167	.606	
Q14	C	.111	.625	+.514	.675	
Q15	C	.000	.625	+.625	.675	
Q16	C	.222	.875	+.653	.344	
Q17	C	.222	.125	−.097	.738	
Q18	C	.000	.750	+.750	.341	
Section D: Generative AI						
Q19	D	.111	.250	+.139	.315	
Q20	D	.000	.500	+.500	.374	
Q21	D	.111	.375	+.264	.838	
Q22	D	.111	.250	+.139	.851	
Q23	D	.111	.250	+.139	.851	
Q24	D	.000	.250	+.250	.851	

Note. LOW = $r_{it} < 0.20$. Pre-test items with difficulty = .000 had all participants select "I don't know," yielding zero variance and no computable correlation. Q3 and Q17 show decreased post-test difficulty, attributable to different item content in the parallel test forms rather than regression effects.

C. Curriculum Overview

This appendix documents the complete structure of the *Pixels to GenAI* curriculum platform. The live site is available at https://burakkagann.github.io/Pixels2GenAI/, and the source repository at https://github.com/burakkagann/Pixels2GenAI.

C.1. Module Summary

Table C.1 lists all fifteen modules with their pedagogical framework assignment, section count, total exercise count, the number of exercises with published content on the live platform at the time of the workshop (February 12, 2026), and whether the module contributed an exercise to the workshop session.

Table C.1.: Summary of the fifteen curriculum modules.

No.	Module Title	Framework	Sec.	Exercises	WS
0	Foundations & Definitions	CDD	4	3 / 4	–
1	Pixel Fundamentals	HOD	3	3 / 9	M1
2	Geometry & Mathematics	HOD	3	10 / 13	M2
3	Transformations & Effects	HOD	4	8 / 17	M3
4	Fractals & Recursion	HOD/CDD	3	4 / 12	M4
5	Simulation & Emergent Behav.	HOD	4	3 / 16	–
6	Noise & Procedural Generation	HOD	4	1 / 15	–
7	Classical Machine Learning	CDD	4	0 / 12	–
8	Animation & Time	HOD	4	3 / 15	–
9	Intro to Neural Networks	CDD	4	3 / 12	M5
10	TouchDesigner Fundamentals	CDD	4	1 / 13	–
11	Interactive Systems	HOD	4	1 / 14	–
12	Generative AI Models	CDD	8	11 / 21	M6
13	AI + TouchDesigner Integration	CDD	4	0 / 12	–
14	Data as Material	HOD	4	0 / 14	–
	Total		**57**	**51 / 199**	

Note. Framework column: HOD = Hands-On Discovery, CDD = Conceptual Deep-Dive. HOD/CDD indicates a hybrid design combining elements of both frameworks. The Exercises

column shows published exercises with content on the live platform out of the total designed exercises. WS = workshop exercise identifier; a dash indicates the module was not tested in the workshop.

C.2. Complete Module Structure

The following listing presents every module, section, and exercise in the curriculum. Exercises marked with ✓ had published content on the live platform at the time of the study; unmarked exercises were structurally present but contained placeholder content only. Exercises marked with **(WS)** were included in the February 12 workshop session.

Module 0: Foundations & Definitions *CDD* — 3/4

- ✓ 0.1 What is Generative Art
- ✓ 0.2 Defining AI, ML, and Algorithms
- 0.3 Images as Data
  - 0.3.1 Creating Images from Arrays
- ✓ 0.4 Setup Environment

Module 1: Pixel Fundamentals *HOD* — 3/9

- 1.1 Grayscale and Color Basics
  - ✓ 1.1.1 Color Basics (RGB) **(WS: M1)**
  - 1.1.2 Color Theory and Color Spaces
- 1.2 Pixel Manipulation Patterns
  - ✓ 1.2.1 Random Patterns
  - ✓ 1.2.2 Cellular Automata **(WS: M2)**
  - 1.2.3 Reaction Diffusion
- 1.3 Structured Compositions
  - 1.3.1 Flags
  - 1.3.2 Repeat Patterns
  - 1.3.3 Truchet Tiles
  - 1.3.4 Wang Tiles

Module 2: Geometry & Mathematics *HOD* — 10/13

- 2.1 Basic Shapes and Primitives
  - ✓ 2.1.1 Lines
  - ✓ 2.1.2 Triangles
  - ✓ 2.1.3 Circles
  - ✓ 2.1.4 Stars
  - 2.1.5 Polygons and Polyhedra
- 2.2 Coordinate Systems and Fields
  - ✓ 2.2.1 Gradient

- ✓ 2.2.2 Spiral
- ✓ 2.2.3 Vector Fields
- ✓ 2.2.4 Distance Fields

- 2.3 Mathematical Art
  - ○ 2.3.1 Lissajous Curves
  - ✓ 2.3.2 Rose Curves
  - ✓ 2.3.3 Harmonograph Simulation
  - ○ 2.3.4 Strange Attractors

Module 3: Transformations & Effects *HOD* — 8/17

- 3.1 Geometric Transformations
  - ✓ 3.1.1 Rotation
  - ✓ 3.1.2 Affine Transformations
  - ✓ 3.1.3 Nonlinear Distortions
  - ✓ 3.1.4 Kaleidoscope Effects
- 3.2 Masking and Compositing
  - ○ 3.2.1 Mask
  - ○ 3.2.2 Meme Generator
  - ○ 3.2.3 Shadow
  - ○ 3.2.4 Blend Modes
- 3.3 Artistic Filters
  - ✓ 3.3.1 Warhol Effect
  - ○ 3.3.2 Puzzle
  - ✓ 3.3.3 Hex Panda
  - ○ 3.3.4 Voronoi Diagrams
  - ✓ 3.3.5 Delaunay Triangulation
- 3.4 Signal Processing
  - ✓ 3.4.1 Convolution **(WS: M3)**
  - ○ 3.4.2 Edge Detection
  - ○ 3.4.3 Contour Lines
  - ○ 3.4.4 Fourier Art

Module 4: Fractals & Recursion *HOD/CDD* — 4/12

- 4.1 Classical Fractals
  - ✓ 4.1.1 Fractal Square **(WS: M4)**
  - ✓ 4.1.2 Dragon Curve
  - ✓ 4.1.3 Mandelbrot Set
  - ○ 4.1.4 Julia Sets
  - ○ 4.1.5 Sierpinski Triangle
- 4.2 Natural Fractals
  - ✓ 4.2.1 Fractal Trees
  - ○ 4.2.2 Lightning Bolts

- ○ 4.2.3 Fractal Landscapes
- ○ 4.2.4 Diffusion-Limited Aggregation

- 4.3 L-Systems
  - ○ 4.3.1 Plant Generation
  - ○ 4.3.2 Koch Snowflake
  - ○ 4.3.3 Penrose Tiling

Module 5: Simulation & Emergent Behavior *HOD* — 3/16

- 5.1 Particle Systems
  - ✓ 5.1.1 Sand Simulation
  - ○ 5.1.2 Vortex
  - ○ 5.1.3 Fireworks Simulation
  - ○ 5.1.4 Fluid Simulation
- 5.2 Flocking and Swarms
  - ✓ 5.2.1 Boids
  - ○ 5.2.2 Fish Schooling
  - ○ 5.2.3 Ant Colony Optimization
- 5.3 Physics Simulations
  - ○ 5.3.1 Bouncing Ball
  - ○ 5.3.2 N-Body Planet Simulation
  - ✓ 5.3.3 Double Pendulum Chaos
  - ○ 5.3.4 Cloth and Rope Simulation
  - ○ 5.3.5 Magnetic Field Visualization
- 5.4 Growth and Morphogenesis
  - ○ 5.4.1 Eden Growth Model
  - ○ 5.4.2 Differential Growth
  - ○ 5.4.3 Space Colonization Algorithm
  - ○ 5.4.4 Turing Patterns

Module 6: Noise & Procedural Generation *HOD* — 1/15

- 6.1 Noise Functions
  - ✓ 6.1.1 Perlin Noise
  - ○ 6.1.2 Simplex Noise
  - ○ 6.1.3 Worley Noise
  - ○ 6.1.4 Colored Noise
- 6.2 Terrain Generation
  - ○ 6.2.1 Height Maps
  - ○ 6.2.2 Erosion Simulation
  - ○ 6.2.3 Cave Generation
  - ○ 6.2.4 Island Generation
- 6.3 Texture Synthesis
  - ○ 6.3.1 Marble and Wood Textures

- 6.3.2 Cloud Generation
- 6.3.3 Abstract Patterns
- 6.3.4 Procedural Materials

- 6.4 Wave Interference Patterns
  - 6.4.1 Moiré Patterns
  - 6.4.2 Wave Interference
  - 6.4.3 Cymatics Visualization

Module 7: Classical Machine Learning *CDD* — 0/12

- 7.1 Clustering and Segmentation
  - 7.1.1 K-Means Clustering
  - 7.1.2 Mean Shift Segmentation
  - 7.1.3 DBSCAN Pattern Detection
- 7.2 Classification and Recognition
  - 7.2.1 Decision Tree Classifier
  - 7.2.2 Random Forests
  - 7.2.3 SVM Style Detection
- 7.3 Dimensionality Reduction
  - 7.3.1 PCA Color Palette
  - 7.3.2 t-SNE Visualization
  - 7.3.3 UMAP Visualizations
- 7.4 Statistical Methods
  - 7.4.1 Monte Carlo Sampling
  - 7.4.2 Markov Chains
  - 7.4.3 Hidden Markov Models

Module 8: Animation & Time *HOD* — 3/15

- 8.1 Animation Fundamentals
  - 8.1.1 Image Transformations
  - 8.1.2 Easing Functions
  - 8.1.3 Interpolation Techniques
  - 8.1.4 Sprite Sheets
- 8.2 Organic Motion
  - 8.2.1 Flower Assembly
  - ✓ 8.2.2 Infinite Blossom
  - 8.2.3 Walk Cycles
  - 8.2.4 Breathing and Pulsing
- 8.3 Cinematic Effects
  - ✓ 8.3.1 Star Wars Titles
  - ✓ 8.3.2 Thank You Animation
  - 8.3.3 Particle Text Reveals
  - 8.3.4 Morphing Transitions

- 8.4 Generative Animation
  - 8.4.1 Music Visualization
  - 8.4.2 Data-Driven Animation
  - 8.4.3 Animated Fractals

Module 9: Introduction to Neural Networks *CDD* — 3/12

- 9.1 Neural Network Fundamentals
  - ✓ 9.1.1 Perceptron from Scratch **(WS: M5)**
  - ✓ 9.1.2 Backpropagation Visualization
  - ✓ 9.1.3 Activation Functions as Art
- 9.2 Network Architectures
  - 9.2.1 Feedforward Networks
  - 9.2.2 Convolutional Networks Visualization
  - 9.2.3 Recurrent Networks and Sequences
- 9.3 Training Dynamics
  - 9.3.1 Loss Landscape Visualization
  - 9.3.2 Gradient Descent Animation
  - 9.3.3 Overfitting and Underfitting Demos
- 9.4 Feature Visualization
  - 9.4.1 DeepDream Implementation
  - 9.4.2 Feature Map Art
  - 9.4.3 Network Attention Visualization

Module 10: TouchDesigner Fundamentals *CDD* — 1/13

- 10.1 TD Environment and Workflow
  - ✓ 10.1.1 Node Networks
  - 10.1.2 Python Integration Basics
  - 10.1.3 Performance Monitoring
- 10.2 Recreating Static Exercises
  - 10.2.1 Core Exercises in Real Time
  - 10.2.2 Boids Flocking in TD
  - 10.2.3 Planet Simulation in TD
  - 10.2.4 Fractals in Real Time
- 10.3 NumPy–TD Pipeline
  - 10.3.1 Script Operators
  - 10.3.2 Array Processing
  - 10.3.3 Custom Components
- 10.4 Interactive Controls
  - 10.4.1 UI Building
  - 10.4.2 Parameter Mapping
  - 10.4.3 Preset Systems

Module 11: Interactive Systems *HOD* — 1/14

- 11.1 Input Devices
  - ○ 11.1.1 Webcam Processing
  - ○ 11.1.2 Audio Reactivity
  - ○ 11.1.3 MIDI and OSC Control
  - ○ 11.1.4 Kinect and Leap Motion
- 11.2 Computer Vision in TD
  - ○ 11.2.1 Motion Detection
  - ○ 11.2.2 Blob Tracking
  - ✓ 11.2.3 Face Detection
  - ○ 11.2.4 Optical Flow
- 11.3 Physical Computing
  - ○ 11.3.1 Arduino Integration
  - ○ 11.3.2 DMX Lighting Control
  - ○ 11.3.3 Projection Mapping Basics
- 11.4 Network Communication
  - ○ 11.4.1 Multi-Machine Setups
  - ○ 11.4.2 WebSocket and WebRTC
  - ○ 11.4.3 Remote Control Interfaces

Module 12: Generative AI Models *CDD* — 11/21

- 12.1 Generative Adversarial Networks
  - ✓ 12.1.1 GAN Architecture
  - ✓ 12.1.2 DCGAN Art
  - ✓ 12.1.3 StyleGAN Exploration
  - ✓ 12.1.4 Pix2Pix Applications
- 12.2 Variational Autoencoders
  - ✓ 12.2.1 Latent Space Exploration
  - ✓ 12.2.2 Interpolation Animations
  - ✓ 12.2.3 Conditional VAEs
- 12.3 Diffusion Models
  - ✓ 12.3.1 DDPM Basics **(WS: M6)**
  - ○ 12.3.2 Stable Diffusion Integration
  - ✓ 12.3.3 ControlNet Guided Generation
- 12.4 Bridging Paradigms
  - ○ 12.4.1 Neural Style Transfer
  - ○ 12.4.2 VQ-VAE and VQ-GAN
- 12.5 Language Models and Art
  - ○ 12.5.1 CLIP Guidance
  - ○ 12.5.2 Prompt Engineering
  - ○ 12.5.3 Text-to-Image Pipelines

- 12.6 Personalization and Efficiency
  - ✓ 12.6.1 DreamBooth Personalization
  - ○ 12.6.2 Latent Consistency Models
- 12.7 Transformer Generation
  - ○ 12.7.1 Taming Transformers
  - ○ 12.7.2 Diffusion Transformer (DiT)
- 12.8 Modern Frontiers
  - ✓ 12.8.1 Flow Matching
  - ○ 12.8.2 Normalizing Flows

Module 13: AI + TouchDesigner Integration *CDD* — 0/12

- 13.1 ML Models in TD
  - ○ 13.1.1 MediaPipe Integration
  - ○ 13.1.2 RunwayML Bridge
  - ○ 13.1.3 ONNX Runtime
- 13.2 Real-Time AI Effects
  - ○ 13.2.1 Style Transfer Live
  - ○ 13.2.2 Real-Time Segmentation
  - ○ 13.2.3 Pose-Driven Effects
- 13.3 Generative Models Live
  - ○ 13.3.1 GAN Inference Optimization
  - ○ 13.3.2 Latent Space Navigation UI
  - ○ 13.3.3 Model Switching Systems
- 13.4 Hybrid Pipelines
  - ○ 13.4.1 Preprocessing in TD
  - ○ 13.4.2 Python ML Processing
  - ○ 13.4.3 Post-Processing Chains

Module 14: Data as Material *HOD* — 0/14

- 14.1 Data Sources
  - ○ 14.1.1 APIs and Scraping
  - ○ 14.1.2 Sensor Networks
  - ○ 14.1.3 Social Media Streams
  - ○ 14.1.4 Environmental Data
- 14.2 Visualization Techniques
  - ○ 14.2.1 Network Graphs
  - ○ 14.2.2 Flow Visualization
  - ○ 14.2.3 Multidimensional Scaling
  - ○ 14.2.4 Time Series Art
- 14.3 Sonification
  - ○ 14.3.1 Data-Sound Mapping
  - ○ 14.3.2 Granular Synthesis

- 14.3.3 Rhythmic Patterns
- 14.4 Physical Data Sculptures
  - 14.4.1 3D Printing Preparation
  - 14.4.2 Laser Cutting Patterns
  - 14.4.3 CNC Toolpaths

C.3. Workshop Module Outputs

Representative visual outputs from the six workshop modules are presented alongside QR codes linking to the live exercise pages in Section 4.6 (Figures 4.1 through 4.7). The complete curriculum platform, including all exercises listed above, is accessible via the QR code in Figure C.1.

https://burakkagann.github.io/Pixels2GenAI/

Figure C.1.: QR code linking to the full *Pixels to GenAI* curriculum platform.

List of Generative AI Tool Usages

In this work, generative artificial intelligence tools were used for the following purpose(s):

Collection of conceptual explications on the topic of this work	☒
Generation of approaches towards scientific methodology or artistic development of this work	☒
Literature search	☒
Literature analysis	☒
Selection of methods and/or models	☐
Generation of code	☒
Generation of visualizations	☒
Generation of musical compositions and/or arrangements	☐
Structuring of the text of this work	☒
Formulation/Editing of the text of this work	☒
Translation of the text of this work	☐
Generation of presentation contents on the basis of this work	☐
Other: Data analysis scripting, statistical verification, and instrument design support	☒

Affidavit

I hereby declare in lieu of an oath that I have independently written this work titled

“Bridging Computational Foundations to Generative AI: A Design-Based Framework for Progressive Creative Coding Education”

and all related parts and that the work is in accordance with the version submitted in digital form. Only the sources and aids expressly named in the work have been used. I have marked as such any ideas taken over directly or in substance.

If generative AI tools were used to write the paper, this has been explicitly listed in the addendum at the end of this submission. Within the core text of my work, I have denoted all relevant contents that were created or paraphrased by generative AI (such as text passages, structures, images, and code or parts thereof) according to the citation style I used for short referencing secondary sources in my submission. In my list of references, each use of generative AI has been listed according to the citation style I used for full references, including the name of the tool, its version number, the date of retrieval, and the prompts used.

I hereby declare that I have not submitted this work in identical or similar form at this or another university previously.

Location, Date	Name / Signature

Legal Caution

SRH Universities require an affidavit of independent authenticity of submitted scientific works in order to be assured that the signatory has generated the latter by him- or herself.

Since the law attributes a special relevance to affidavits and arising consequences thereof, submission of a false affidavit entails criminal punishment. Deliberately (and therefore knowingly) submitting a false affidavit may be punishable by up to three years imprisonment or a fine. Negligently submitting an affidavit (i.e. submitting despite having been able to recognize that the affidavit does not match factual truths) may be punishable by up to one year imprisonment or a fine.

The punitive frameworks are set forth in Section 156 of the German Penal Code (*falsche Versicherungen an Eides Statt*), as well as in Section 161 of the German Penal Code (*fahrlässiger Falscheid, fahrlässige falsche Versicherung an Eides Statt*).

> **Section 156 StGB: False declaration in lieu of oath**
> Whoever falsely makes a declaration in lieu of an oath before an authority which is competent to administer such declarations or falsely testifies whilst referring to such a declaration incurs a penalty of imprisonment for a term not exceeding three years or a fine.
>
> **Section 161 StGB: Negligent false oath; negligent false declaration in lieu of oath**
> (1) Whoever commits one of the offences referred to in sections 154 to 156 by negligence incurs a penalty of imprisonment for a term not exceeding one year or a fine.
> (2) No penalty is incurred if the offender corrects the false statement in time. The provisions of section 158 (2) and (3) apply accordingly.

I hereby acknowledge the above

Location, Date	Name Signature