NEURAL IMPRINT · WHITEPAPER V2

Next-Generation NI: A Vision, Philosophy, and Technical Path for Continual Learning

Whitepaper v2 · AtomGradient · September 17, 2026

Date 2026-09-17 Version v2 Language English · 中文

This whitepaper presents AtomGradient's research position on continual learning, the algorithmic and engineering foundations we have established, and the direction of the next generation of Neural Imprint.

We see continual learning as a practical problem that spans algorithms, models, and device operation. Our understanding of intelligence guides the research; algorithm design provides the learning mechanisms; engineering and experiments put those mechanisms to the test in the real world.

AtomGradient · 质子梯度（北京）科技有限公司
Online edition and corrections: https://atomgradient.github.io/whitepapers/neural-imprint/en/

Contents

1. NI's Goal Has Always Been Continual Learning

Neural Imprint (NI) is a continual learning algorithm and algorithmic architecture developed by AtomGradient. Around NI, we have built a supporting technology stack that covers model optimization, inference runtimes, learning-state management, tool integration, developer interfaces, and cross-device collaboration. Our goal is for models running on users' devices to build understanding from new records, interactions, and action feedback, and to use what they learn in the next task.

The first generation has already established a foundation for this on phones: a model first processes information about the user and instructions for available tools, preserves the internal computational state formed in doing so, and carries that state into subsequent sessions. As new records and user corrections accumulate, the existing workflow supports incremental learning, updating and activating new learning states. Building on this foundation, the next generation will investigate finer-grained updates, longer-term capability retention, and continuity of learning across devices.

For users, this means the model can carry its understanding of their characteristics and tool usage into later sessions, reducing the need to explain the same things repeatedly. When a question requires precise records, the model can use connected tools to look them up.

NI has pursued continual learning from its first generation. We are working toward the same goal, developing the mechanisms already established into a more complete learning loop.

Memory, representations, the structures that hold state, tool use, and adjustments to inference all serve this loop. The central question is whether experience leaves the model better able to handle what comes next—and whether it can keep revising its understanding as new evidence arrives.

1.1 Enabling Agents to Keep Growing After Deployment

Foundation models have given agents strong capabilities in language understanding, perception, and task execution. Yet performing well on a single task and continuing to grow through long-term use are still different things.

A model may understand a new piece of information today and still need us to explain it again tomorrow. It may accept a correction now without changing how it handles a similar situation later. This gap between understanding and lasting accumulation is precisely what continual learning seeks to address.

Test-Time Continual Learning (TTCL) focuses on agents continuing to acquire, consolidate, and refine capabilities after deployment. Our position is that growth should extend throughout a model's useful life, rather than take place only before delivery. Here, “test time” primarily refers to the period of actual use.

We aim to make that growth happen on users' devices and in their environments. Models should form new understanding from observations, interactions, action outcomes, and corrections; retain capabilities that remain useful; and apply what they learn to situations they have not encountered before. Everyday updates should build on prior learning at an affordable incremental cost.

This requires us to study understanding, prediction, action, and long-term reliability together. The value of continual learning lies in experience continually becoming a capability for facing the future.

1.2 What Separates Temporary Adaptation from Sustained Growth?

What we find most unsatisfying about today's large models is how difficult it is for understanding formed during one interaction to become a capability they retain later. In-context learning shows that models can adapt to a task using newly supplied examples and definitions. Continual learning seeks to make that adaptation persist.

The real distinction is this: after understanding something once, does the model need to be reminded from the beginning next time? Can a correction received today remain effective in relevant situations? If circumstances change, can the model recognize that an earlier understanding no longer applies?

We use “learning continuity” to describe this goal: turning adaptation within the current task into changes that can be used, tested, and revised across tasks.

1.3 The Research Question Running Through This Paper

> How can NI use sustainable incremental mechanisms to help an agent organize, compress, and consolidate a continuing stream of experiences into capabilities it can use, revise, and transfer—while maintaining learning continuity under limited resources and across heterogeneous environments?

We choose how to carry learning based on what learning needs to achieve. Changes may reside in model parameters, internal state, or both; different timescales may also call for different update methods. A mechanism's value depends on how change happens, why it is useful, and

how it persists.

Tools provide observations, actions, and authoritative facts. NI further asks how models can form reusable understanding and strategies from the experience of using those tools.

1.4 Intelligence Goes to the Data

We once posed a question: does intelligence go to the data, or does the data go to intelligence?

AtomGradient's answer is: intelligence goes to the data.

This determines where we develop learning capabilities: we bring models into the devices and environments where data originates, so that understanding, responses, and accumulated learning take place where personal experiences occur.

Closely connected to this is a research position we have always held: **the device is the agent.** Using each device's own computing capabilities, we bring together models, sensory inputs, available actions, and persistent learning state, so that observation, understanding, action, and learning happen within the device's environment.

The device thus becomes the entity in which experiences occur, capabilities operate, and learning accumulates. Local computation allows it to process its own observations, use its own tools, and update its understanding from new feedback. Continual learning gains a concrete home: a device that remains with its user over time and keeps growing through use.

Bringing intelligence to the data also shapes the relationship between models and people. Real life is not a dataset submitted once and for all. Habits change, environments change, tools change, and a person's understanding of the same issue can change too. Intelligence that can keep learning where these changes occur has a chance to participate in a lasting relationship, rather than merely process a series of isolated requests.

This also concerns control. Users should decide which experiences are learned from, which understandings should be revised, and which states may be saved, deleted, or transferred. Learning states need explicit privacy protection, just as raw data does.

The same principle applies to cross-device learning. We want learning to continue across devices with different computing and sensing capabilities, while raw data remains on users' devices. The movement of learning states must respect explicit compatibility and authorization boundaries.

Choosing to bring intelligence to the data means confronting real devices' memory, power, bandwidth, and operating conditions from the outset. These constraints directly shape the design of the learning mechanism.

2. Turning the “Paper Notes” Metaphor into a Learning Problem

2.1 Paper Notes as an Entry Point for Experience

We once imagined this through a multidimensional space: an agent lives inside it, and paper notes can be attached to every wall. Incoming records are the notes. One experience may relate to several walls, and multiple notes may share the same representational location. When a question arrives later, the agent can find the relevant content and express it again.

In this metaphor, a “note” corresponds to an observation or a record, and a “wall” to the internal space in which a model organizes information. “Attaching” means allowing a new experience to change internal state; “finding” means using the current question to evoke relevant information; “layering notes” means accommodating different experiences within shared structure.

We chose paper notes because of the forms in which information can be presented. Text can be a record, an image a single view, video a sequence of frames, and audio a transcription. The metaphor turns a continuous stream of information into units that can be processed.

A note is presented in two dimensions, but the relationships it carries can be multidimensional. A record may connect time, objects, goals, context, and outcomes at once. When video is examined frame by frame, the order of actions matters; when audio becomes text, information such as tone and pauses may be lost. What we seek to learn from is interconnected experience, not just isolated units of data.

The “walls” invite us to think about representation spaces: could the same experience be understood and used through different relationships, rather than assigned to one fixed drawer? We are more interested in whether the model can learn this organization than in having developers predefine a classification rule for every situation.

2.2 What Does It Mean to Superimpose Notes?

The most interesting part of this idea is that experiences need not each occupy an entirely separate slot. They may share some structure while retaining the conditions that matter to each one.

Shared representations create an opportunity for compression and a risk of interference. Two experiences that look similar may require opposite actions because a crucial condition differs. Two experiences that look different may reflect the same underlying pattern. Preserving these distinctions is closer to the problem of learning than simply adding storage.

The key to “layering” is therefore selective sharing: identify reusable structure when writing, preserve the conditions that determine outcomes, and understand which set of relationships a current cue refers to when reading.

Expressing an internal representation in language makes retained information usable again. The model should also distinguish recollection, inference, and the unknown: what comes from a particular experience, what is a judgment based on a pattern, and what still lacks evidence.

2.3 Remembering Experiences, Understanding Patterns, and Developing Capabilities

This involves three connected levels. **Remembering an experience** means knowing what happened on a particular occasion. **Understanding a pattern** means recognizing possible relationships between conditions and consequences. **Developing a capability** means using that understanding to choose and carry out appropriate actions in a new situation.

To explain this progression clearly, we need to distinguish episodic memory, semantic memory, and what is commonly called “factual memory.” Tulving's research on episodic and semantic memory helps separate recollection of specific experiences from possession of knowledge. We use this functional distinction to design learning tasks for models, focusing on whether information from experience can be retained and used.

Concept	What question does it answer?	Example in an agent
Episodic memory	What happened in a particular context?	Remembering the goal of an attempt, its conditions, the action taken, and its consequences, and recalling the episode when relevant cues appear
Semantic memory	What is already known, and how can it be understood?	Knowing what a tool means, how a class of environments works, or what has been learned about personal preferences, without having to replay the original learning episode
Factual memory	What factual content has been retained?	When an event occurred, an object's attributes, or a learned definition; factual content can occur in episodic memory or become semantic knowledge

These are explanations of three common terms, not three mutually exclusive compartments. Episodic and semantic memory distinguish how information is organized and used; “factual” describes its content. Semantic memory includes facts and concepts, while episodic memory also contains facts about events. We further use “exact factual recall” for the task of answering questions about the details of original records, and discuss it separately from forming understanding from experience.

NI asks which details determine the meaning of an experience and which understandings are worth using again. For questions that require exact amounts, dates, original wording, or the latest status, a model can access authoritative records through tools. Learning retains the structures that help it understand situations, predict consequences, and choose actions. Not memorizing every raw fact is therefore compatible with retaining useful episodic and semantic knowledge.

For example, an agent's failure in a game may be retained as an episode with conditions and consequences. Comparing related experiences may help the model form an understanding of how the environment works. In an entirely new situation, it can then use the current conditions to judge whether that understanding applies. This last step is adaptive reasoning: using what has been learned to address a new problem, and revising a judgment when the evidence changes.

Memory supplies material for reasoning; reasoning puts that material to work in the current problem; learning then incorporates new feedback. **Continual learning aims to make this loop continually develop capabilities.** Episodes need not be preserved word for word, and generalizations should retain their conditions of applicability, so the model can benefit from the past without being trapped by it.

2.4 Learning Is Compression, but Smaller Is Not Always Better

We have long regarded compression as essential to understanding learning. People cannot retain every detail of every fact, yet experience can give rise to habits, ways of judging, and skills. We want models to preserve structures that matter for the future, rather than merely accumulate the surface forms of the past.

By compression, we mean preserving meaning and function: discarding irrelevant differences while retaining information that affects understanding, prediction, and action. Its purpose is to make experience more useful, not simply to minimize size. Information bottleneck theory addresses this trade-off: how much task-relevant information to retain in a limited internal representation while compressing other details.

An added difficulty in continual learning is that future questions are not fully known. A detail that seems irrelevant today may determine tomorrow's judgment. We need to study not only how to generalize, but also the scope and uncertainty of those generalizations, and how to reorganize understanding when exceptions appear.

Selective forgetting is a necessary trade-off under finite resources. Catastrophic forgetting is the uncontrolled loss of still-important capabilities while learning something new. We aim to take in new experience while preserving understanding and skills that remain valid.

3. The Core of Next-Generation NI: Letting Experience Change State the Model Can Use

3.1 Observation, Action, and Updating Should Form a Loop

We organize learning as a continuous feedback loop: existing capabilities and current experience jointly determine how to update; the updated model and state then participate in the next act of understanding and action.

```
Observe + use existing state → Understand → Act
                ↑                           ↓
Retain, consolidate, revise ← Experience + feedback
```

Experience here is more than receiving a passage of text. It can include what was done, under which conditions, what happened, and where an earlier prediction was wrong.

An agent can observe actively, explore to reduce uncertainty, and revisit a failure. Actions change the experiences that become available next, and those experiences should change the next action.

This loop must also distinguish two kinds of change: tracking the current situation and forming understanding that persists across experiences. Knowing one's current location and learning “how to act under these conditions” both require internal state, but they are not the same learning outcome.

3.2 Incremental Learning: Building on What Has Already Been Learned

We have a straightforward requirement for continual learning: each new batch of experience should build on what has already been learned.

Incremental learning emphasizes absorbing new information using prior learning. Continual learning also faces long-term retention, correction, transfer, and environmental change; online learning places greater emphasis on processing data as it arrives. These concepts overlap in research. What matters is specifying what is updated, when, and at what cost.

The first generation of NI already supports incremental learning. The on-device application accumulates new records and user corrections, and prompts the user to learn again when update conditions are met. The user decides when to begin.

Once the process starts, the system identifies unchanged records by their content and reuses their existing internal representations, computing new representations only for new or changed content. RPP then combines old and new representations to derive personal characteristics, generate a new learning state, save it, and activate it for subsequent inference. This workflow—from incoming data to an updated learning artifact and its use in inference—has already been validated on physical devices.

Source: AtomGradient internal research and test records.

The current mechanism combines incremental representation extraction with periodic analysis across the accumulated material: unchanged records do not need their representations extracted again, while personal-characteristic analysis and state generation use old and new material together. Building on this operating learning path, the next generation will investigate finer-grained updates, so that everyday learning costs track the amount of new information more closely, while strengthening long-term consolidation and retention of earlier capabilities.

Review and reorganization still matter. People also revisit, generalize from, and reinterpret past experiences. We advocate combining everyday incremental updates with periodic consolidation: the former maintains continuity, while the latter turns accumulated experience into more stable, reusable structure.

3.3 The Model Is the Product: The Model Carries What Has Been Learned

NI has a clear design goal: the outcome of continual learning is a model. Changes caused by experience should be internalized, retained in parameters or learning state, and used in subsequent understanding, prediction, and action.

The model is the product: product value accumulates as the model learns. The value users build over time comes not only from general capabilities present at delivery, but also from the understanding, judgment, and ways of acting that the model develops later. We want this growth to become something the model itself can carry.

Putting every record in a database and retrieving it for the model when needed is a useful technical approach. Its primary question, however, is how to retrieve external content. NI asks a further question: how do those experiences change the way the model understands and uses information?

The unit of transfer should therefore be a complete model that carries its learning: both the parameters that perform computation and the internal learning states that hold acquired understanding and participate in later computation. If learning changes weights, the updated

weights must be retained. If fast weights or other neural memory states carry the changes, those states are part of the learning artifact. Whether they are stored in one file or several is a matter of implementation and delivery.

We want the model to carry its understanding of past situations, its knowledge, and its adaptive reasoning capabilities into another compatible device, without separately bringing raw experience archives or an external retrieval store to reconstruct that understanding. Further consolidating learning into a unified set of model parameters is one path worth exploring. In choosing how learning is carried, we focus on the relationship between usable capabilities, continued updating, and device costs.

We judge internalization by function: has experience changed the model's subsequent understanding, prediction, and action, and can that change remain useful and open to correction? Devices continue to provide computation, perception, and action interfaces; tools continue to provide current information and authoritative facts. The model enters its new environment with the understanding it has already acquired.

3.4 Restoring Learning, Not Just Restoring Use

Long-term growth means that after a restart, a model can both use what it has learned and keep learning through the same mechanism.

Restoring learning requires preserving both readable learning outcomes and the information needed to update them. An associative mapping, for example, may support reading, while further updates may depend on accumulated statistics or other auxiliary state. Learning continuity requires these dependencies to be part of the design.

This gives transfer two requirements: first, restore acquired capabilities with the model; then, allow further growth to continue from the same point. The next generation needs to investigate which information must travel with the model, which can be replaced by consolidated state, and how to absorb new experience without repeatedly reading the entire history.

During learning, raw material and historical representations may remain on the device at the user's discretion, for review, correction, or periodic reorganization. These should be distinguished from the learning artifacts that must be carried during transfer. We aim to progressively reduce the dependence of capability use and everyday updates on raw history, allowing the model to keep growing from the understanding it has already formed.

After restoration, the model should continue from where it left off: using prior understanding, accepting new evidence, and allowing old and new state to evolve coherently.

4. The Technology Stack We Have Built Around NI

Continual learning requires algorithms, models, and devices to work together. Starting from the learning objective, we define the required capabilities from the top down, then build the supporting technology stack from the bottom up. NI provides the continual learning algorithm and architecture; the supporting engineering brings these mechanisms onto real devices and enables their operation, updating, validation, and use.

The stack connects the following areas of work:

Area	Capabilities we have established	Role in continual learning
Experience representations and learning mechanisms	RPP, on-device incremental learning, direction construction, and activation-intervention interfaces	Form usable understanding from records and corrections, allowing changes to affect subsequent inference
Model analysis and optimization	Activation analysis, pruning and quantization, before-and-after comparisons, and model and application export	Study model capacity and computation trade-offs under device constraints, and bring the results into use
Inference runtime and resource management	DSR, FrogJump, cache quantization, device budgets, execution scheduling, and multimodal state management	Determine which states are retained and which computations are performed, providing the operating foundation for long-term interaction
Learning-state lifecycle	Generation, saving, activation, restoration, removal, and compatibility checking of learning artifacts	Make learning usable across sessions, with consistent updating and restoration
Tool integration and developer interfaces	Support for tool contracts, execution of tool calls, and interfaces that connect models, tools, and learning workflows to device applications	Connect a model's understanding to the information a device can obtain and the actions it can perform
Cross-device collaboration and state management	Inter-device communication, compatibility checks, and mechanisms for transferring and restoring learning state	Let prior learning remain usable across authorized, compatible devices

Source: AtomGradient internal research.

These efforts serve a common objective: devices should be able to use models over long periods, take in new experiences, update what has been learned, and use those updates in later tasks. We first describe the operating requirements and core mechanisms, then present representative measurements showing what the engineering achieves in practice.

4.1 Local Continual Learning Requires Models That Can Keep Running

Bringing intelligence to the data first requires models to perform perception, inference, and interaction on the devices where that data resides. Continual learning depends on recurring observation, action, and feedback. Sustained model operation on real devices is therefore our starting point.

The difficulties on devices are closely interdependent: model weights, session state, and temporary computation share limited memory; new images and speech keep arriving; and different chips incur different execution and data-movement costs. Addressing these issues requires understanding a model's computational structure and designing state management, memory allocation, and execution scheduling around it.

An inference runtime is the software that arranges model computation, memory use, and state management. By designing model optimization and the runtime together, we have made concrete progress in sustained inference, session-state preservation, and speech execution on non-Apple platforms. Building on these results, next-generation NI will further investigate the computation and state requirements of learning updates, so that models can acquire new capabilities through continued interaction.

4.2 Core Mechanisms in Model Optimization and the Runtime

We break model operation into several problems that must be addressed together: which historical states are worth keeping, which computations can be omitted, how limited memory should be allocated, how context can survive a change in perceptual workload, and how optimized models can be put to use on devices. We have developed the following mechanisms and tools around these questions.

DSR (Dual-Sparse Retention): choosing which history to retain within a limited cache.

In a long conversation, the attention cache grows as inputs and outputs accumulate. Also called the KV cache, it stores internal representations that later computation will query. DSR divides the available space among a small number of initial positions, recent content, and historical content with higher attention scores. Retention budgets also depend on layer position and scenario configuration.

Importance is estimated here from attention scores produced during model execution, and those scores are updated as computation continues. DSR uses them to select states for retention and periodically compacts the cache, bringing the preservation of history within an explicit resource budget. It turns “what should limited memory be used for?” into an executable runtime rule and is one of our core mechanisms for supporting extended interaction.

FrogJump: using model structure to organize a leaner computation path.

This mechanism targets hybrid architectures containing both recurrent and attention layers: recurrent layers continually integrate prior information, while attention layers read the information they need from retained representations.

FrogJump generates a layer-skipping plan from the structure of a supported model. During execution, it bypasses selected intermediate recurrent layers while preserving every full-attention layer and the recurrent layer immediately before it. The model can thus process input through a path that requires less computation.

We have implemented the skipping plan, integrated it into inference execution, and added corresponding structural checks and tests. The current path is explicitly enabled in the non-thinking mode of supported models. FrogJump makes computation trade-offs configurable in the execution path; output quality, task performance, and actual running costs are the outcomes to examine together when evaluating such a path.

Cache quantization and hybrid-state management: considering both how much to retain and how to store it.

DSR selects states; cache quantization determines their numerical representation. Quantization stores values with fewer bits, reducing cache memory while introducing numerical error that needs to be evaluated. We design quantization formats, cache budgets, and execution paths together, so that storage representation works with model computation.

Hybrid architectures also require separate management of different kinds of state: attention caches retain representations for later queries, while recurrent states integrate prior information into fixed-shape numerical state. Our runtime handles their allocation, updating, and restoration separately, allowing the different patterns of state growth to be managed independently.

Device budgets and execution scheduling: adapting computation to actual operating conditions.

We plan resources using available device memory, model footprint, bandwidth information, and execution capabilities. We reserve space for temporary computation, then allocate cache budgets and input-processing batch sizes. As context grows, execution scheduling can also adjust the amount of computation submitted in each batch to control transient pressure.

This connects model structure with device conditions. The same model can use different cache sizes, computation batches, and submission schedules on different devices, providing a mode of sustained interaction suited to the hardware available.

Multimodal state continuity: preserving what needs to continue when computational tasks change.

When visual processing and language decoding take turns using resources, the runtime manages weights and session state separately. During chunked speech generation, it carries forward the internal state needed for subsequent output. To support these differences, we have implemented staged resource allocation and state-preservation paths, allowing a new observation or the next output chunk to continue from prior computation.

Model analysis, optimization, and delivery: bringing research changes into practical use.

We have also established workflows for activation analysis, pruning and quantization, reloading and quality checking after optimization, and exporting models and applications. Activation analysis helps us observe how computation is used. Pruning and quantization offer different choices of capacity and numerical representation. Comparative checks help assess the effects of those changes.

Supporting developer interfaces connect model execution, tool execution, and learning-state management to device applications. This links model optimization, learning mechanisms, and device operation: research changes can be loaded, tested, and used, while results from devices can inform the next round of research.

Runtime configurations also respect state-compatibility requirements. Long conversations can use budgeted cache management; restoring a captured, complete learning state uses a configuration compatible with the one that generated it, keeping state layout and execution consistent.

Source: AtomGradient internal implementation and research records.

Together, these mechanisms determine how a model uses limited computation and storage. The learning mechanisms that follow address how new experiences enter model state that can be retained and used.

4.3 RPP

RPP extracts relatively stable signals of personal characteristics from multiple records. It takes the internal numerical values produced when a model processes those records and outputs a set of numerical directions for analyzing personal characteristics, helping identify structures that recur across records.

Specifically, RPP first uses a predefined set of general feature directions as references, separates out their corresponding components in the records, and analyzes the remaining structure. These general directions describe common features and serve as shared references across users. Weighted principal component analysis and stability checks then produce directions for extracting personal characteristics.

“Activations” here are the internal numerical representations formed as a model processes input; a “direction” describes a particular kind of variation in those representations. Principal component analysis finds more prominent patterns of variation, while stability checks examine how readily those patterns change with the sample.

RPP lets us examine personally relevant structure within a model's representation space, rather than merely produce a surface-level summary of the original text. The next generation can build on this foundation while also investigating how representations, analytical objectives, and subsequent use should evolve.

In the current workflow, these directions help select representative records. A model then uses those records to summarize personal characteristics. The resulting text, together with tool instructions, supplies the input for generating the reusable state described in the next section.

Source: AtomGradient internal research.

Building on this foundation, the next generation will further investigate representations of time, conditions, and action consequences, so that the structures identified can also participate in later judgments and predictions. This is an important connection between personally relevant structure and richer learning capabilities.

4.4 How Does Learning State Enter Inference?

The model first reads text describing user characteristics and available tools. We save the internal computational state formed after that text has been processed. A later session can continue from this state to handle the current question, reusing computation that has already been performed.

The first generation of NI already supports generating, updating, saving, restoring, and removing these states. Technically, the text processed first is called a “prefix”; its “tool contracts” describe the tools' functions, parameters, and usage.

This allows previously formed computational state to be reused across sessions, with subsequent inputs processed on that foundation. On the next learning run, updated personal characteristics and tool instructions generate a new state, replacing the version previously used for inference.

Later sessions can restore the new state and continue working. The next generation will investigate finer-grained state updates, bringing the writing of new understanding, correction of prior understanding, and long-term consolidation into closer coordination.

Different model architectures provide different retention and integration mechanisms. Some states grow with the input sequence and support later attention queries; others recurrently integrate prior information into fixed-shape state. Understanding these differences helps us choose suitable structures for different capacity, reading, and updating requirements.

4.5 Tool Learning and Personal State in Inference

Tool learning is an established foundation of first-generation NI. Tool contracts describe what tools can do, which parameters they accept, and how to use them. The model processes these contracts and forms a state. Once that state is restored, it can generate tool calls based on the current question, and the runtime executes them.

For example, when a user asks how much they spent in a particular month, the model can retrieve the data through a connected spending-record query tool.

In this workflow, the model learns the tool contracts during state construction. Once the actual tools have been implemented and registered, the resulting state can be reused during operation, without reinserting the same tool instructions into every request.

We have validated the end-to-end on-device workflows for answering questions about personal characteristics and making the corresponding tool calls. This includes state restoration, generating calls from the current question, and runtime execution of the tools.

Source: AtomGradient internal test records.

Personal state allows the model to adjust its answers using established preferences and patterns, making it closer to semantic memory as discussed earlier. Starting from this foundation, the next generation will further investigate retaining specific experiences along with their time, context, and consequences, and using that understanding to reason when conditions change.

The next generation must also move from knowing tool contracts to improving strategies through experience: when a call is worthwhile, how to respond to failure, and how to adapt when a protocol changes. A successful tool call is a foundation for this path, not its endpoint.

4.6 Activation Steering and Directional Steering

Activation Steering adjusts the internal numerical values a model is using while generating an answer, influencing what it generates next. These runtime values are called “activations.” This intervention can be used without changing model weights.

Anthropic's Golden Gate Bridge experiment provides an intuitive example: amplifying a feature associated with the Golden Gate Bridge made the model mention it even in unrelated conversations. This demonstrates the direct effect of internal activations on generated output and highlights the importance of intervention strength and context.

Our current implementation includes two direction-processing paths. One uses RPP-generated directions to examine the relationship between current input and existing structure, and prepares the numerical data needed for intervention. The other uses activation differences before and after a correction to construct directions that influence subsequent computation; we call it **Directional Steering**.

Both paths concern internal activations. They differ in how directions are obtained, when they are used, and how they enter computation. A direction is a coordinated set of numerical adjustments. Directional Steering is a specific implementation of activation intervention.

In the current application workflow, RPP directions are used to examine the relationship between input and personally relevant structure. For interventions using correction-derived directions, loading, reloading, and removal have been verified in dedicated runs. This provides a foundation for further research into direction selection, context matching, and intervention effects.

Source: AtomGradient internal research.

The next generation will focus on how directions correspond to learning objectives, how to choose the contexts for intervention, and how interventions interact during continual updating.

4.7 Representative Engineering Results and Measurements

We have selected three cases with concrete execution records to illustrate sustained operation, multimodal state continuity, and execution efficiency. The long-conversation case combines cache quantization, memory budgets, dynamic scheduling, and bounded cache retention. The multi-image and speech cases demonstrate improvements in state management and computational organization, respectively.

Source for the measurements in this section: AtomGradient internal test records.

Controlling cache and computation costs in long conversations so that large models can keep running on phones.

Long conversations continually increase cache and computational pressure. Some model layers integrate history into fixed-size state, while others retain an attention cache for later queries. We manage these two kinds of state separately, store caches in more compact numerical formats, set memory budgets, and adjust computation batch sizes as context grows. This allows the runtime to keep generating within the device's memory constraints.

In sustained tests on two phones, a 9-billion-parameter model completed 200 rounds of questions and answers within a single session on each device. Each generated 204,800 tokens in total, with peak memory of approximately 5.5 GB. A token is a basic unit of text processing in a model; the observation here is whether the phones can keep completing one round of generation after another.

Test conditions: Qwen3.5-9B-4bit on iPhone Air and iPhone 17 Pro, with exactly 1,024 tokens generated per round, continuous external power, and active cooling.

This work brings state growth and execution pressure in long conversations within an explicit budget, providing an operational device foundation for sustained interaction.

Separating weight loading from session-state management so new images can enter an existing context.

Image processing requires additional memory. Releasing an entire language-model session also discards the state already formed, forcing history to be recomputed later. We separate the lifecycle of model weights from session state: temporarily unload decoder weights while preserving attention and recurrent states, then reload the weights after visual encoding and process only the new image and added text.

In a controlled comparison of multi-image conversations, the wait from submitting the second image to the start of the model's answer fell from 8.15 seconds to 6.64 seconds—a reduction of about 1.5 seconds—while the existing session state was preserved. Keeping that state added approximately 190 MB to peak memory.

Test conditions: an iPhone Air running a Qwen3.5-9B multi-image workload; the state-preservation mechanism is an experimental path, disabled by default and enabled for this test.

This change addresses the need to recompute history when switching perceptual workloads. The device can free resources for a new observation while retaining the session state already formed. It also illustrates an important principle in our runtime design for continual learning: resources can be rearranged, while state should have independent mechanisms for preservation and restoration.

Reorganizing speech-predictor execution to improve computational efficiency on a non-Apple platform.

The speech predictor generates the encoded information needed for synthesized audio step by step. Each step needs only its corresponding output component, or “output head.” We split the computation shared across steps—the shared trunk—and the output heads into separate

execution units. Keeping the original weight values and the trunk's state-update structure, we run only the fixed-weight output head needed at that step and pass the trunk's output directly to it.

On an older chip on a non-Apple platform, the median time for the predictor's fixed 15-step computation fell from 40.05 milliseconds to 32.28 milliseconds. In a separate comparison using three streaming inputs, the generated audio was byte-for-byte identical before and after optimization.

Test conditions: identical inputs, with the original and optimized paths run alternately over 10 rounds. These timings measure predictor computation, including invocation and waiting.

In the complete speech pipeline, we also implemented stateful chunked generation, allowing the device to provide playable audio while synthesis continues. In a local test with 10 sessions, the median wait from the service receiving text to emitting the first audio chunk was approximately 0.37 seconds. A separate 30-minute synthesis test completed 357 utterances with 0 failures.

This optimization retains the existing weights and reduces computation time by changing how execution is organized. It illustrates our approach to model optimization on heterogeneous devices: combine an understanding of model structure with the platform's execution characteristics to organize the computation that actually needs to take place.

From resource management in long conversations, through continuity of state during multimodal transitions, to reorganized speech generation, we have established concrete engineering capabilities for keeping models resident and interactions ongoing. Together, they support NI's research direction: enabling the observation, inference, and state accumulation on which learning depends to continue on users' devices.

4.8 From Existing Foundations to the Next Generation

We already have concrete foundations in device operation, representation analysis, on-device incremental learning, tool-contract support, personal state in inference, direction processing, and state restoration. New records and corrections can enter the learning workflow, and updated learning artifacts can be saved, activated, and used in subsequent inference.

The next generation will evolve from this operating learning loop, expanding methods for representation, updating, and consolidation. What can accumulate will extend from personal characteristics and tool usage to contextual understanding, action consequences, and transferable strategies.

5. Why Do Language, World Models, and Agents Come Together Here?

5.1 Understanding Not Only What Happened, but What Could Happen

Language models are adept at working with structure, knowledge, and expression in language. World models direct our attention to another question: if a particular action is taken in the current situation, how might the environment change?

The original World Models work connects perceptual representations, temporal prediction, and control. Dreamer further demonstrates a route to learning behavior through imagined trajectories based on internal predictions. What matters to us is the underlying principle: representations should serve consequence prediction and decision-making.

For NI, this suggests an important direction. The value of experience lies not only in being able to answer “what happened?” later, but also in helping us judge “under these conditions, what might happen next?”

Predicting consequences requires representations to preserve the relationships among actions, temporal order, and environmental conditions. We want models to identify which conditions an outcome depends on, then correct internal predictions using real feedback to reduce error accumulation across successive imagined steps.

5.2 The Device Is the Agent: Models Enter the Environment Through Devices

In our research, an agent is the form a model takes when it enters a loop of goals, observations, actions, and feedback. The device provides local computation, sensing and interaction interfaces, and executable actions. The model provides understanding, prediction, and action selection. NI allows changes formed through experience to be retained, updated, and used again. Together, they enable the device to operate as an agent in its own environment.

This also explains why we place computing capability at the foundation of the agent concept: a device must be able to process information locally, run models, and manage learning state to exercise judgment and learn through continued interaction. Phones, laptops, and other heterogeneous devices can differ in computing power, sensing capabilities, and available actions. Our research aims to let them take on different roles according to their capabilities, collaborate with user authorization, and carry prior learning forward.

Such an agent can actively obtain information, use tools to change its environment, and recall earlier attempts.

This changes the learning problem. In passive reading, what comes next is largely determined externally. In active interaction, the agent's choices affect its future data. It can keep doing familiar things, or incur a reasonable cost to resolve an uncertain judgment.

Autonomous learning therefore concerns not only how to write new content, but also what to observe next. Learning to ask valuable questions, assess the reliability of feedback, and recognize what is not yet known can themselves become capabilities.

5.3 Learning Through Games: How Does Experience Become Capability?

We find games a useful way to understand this. An agent enters an unfamiliar environment, tries actions, observes results, gradually discovers which conditions matter, and develops reusable ways of judging. Rather than memorizing the record of every game, experience changes how it approaches the next one.

In this setting, learning appears as continuing change: earlier experience affects later strategies, successful methods are consolidated, mistaken understanding is corrected, and behavior can readjust when the environment changes. We will observe and test the accumulation of capabilities through these changes.

Completing the current task and becoming more capable through the current task are related but distinct goals. Continual learning brings the second goal into system design and evaluation as well.

5.4 Could These Functions Be Unified in One Model?

Unifying language expression, environment prediction, and action selection is an important research direction for us. These functions could share internal representations, allowing a model's understanding of the world to support expression, internal simulation, and action together.

WorldVLA uses a unified autoregressive model for action generation and future visual prediction, offering a concrete reference for this kind of functional unification. Our next question is how continual updating and experience consolidation can become part of such a unified structure.

Our deeper aspiration is for the model's explanations, predictions, and actions to rest on a coherent understanding. Shared representations could reduce fragmentation and conversion between components. They also require us to address interactions among functions, including how errors are detected, contained, and corrected.

We therefore treat architectural unification and continual updating as connected research questions: the former joins understanding, prediction, and action; the latter allows those capabilities to keep growing with new experience.

5.5 What Role Does Each Part Play?

Language and multimodal capabilities help a model understand and express information. World modeling helps it predict consequences. NI studies how experience continually changes model state that can be used. An agent places these capabilities in real interaction, while evaluation environments test what those changes actually accomplish.

These responsibilities can be organized in different architectures or gradually unified in a single model. The key relationship remains clear: the model changes internally, the agent gains experience in the environment, and evaluation tests the results through independent tasks.

6. Technical Methods and Philosophical Insights: How Should Learning Mechanisms Be Designed?

We design methods around concrete learning questions: how experience is represented, how it causes updates, how it is used again, and how the whole process continues over time.

6.1 Start with Representations That Preserve the Distinctions That Matter

If two events with entirely different consequences are represented by nearly identical states, even a sophisticated reading mechanism may confuse them. If different expressions of the same thing lie far apart, what is written and what is queried may fail to match.

Representation is therefore more than converting input into a string of numbers. It determines which differences survive and which relationships can readily be used. For continual learning, representations should support recall, judgment, and consequence prediction together, rather than merely place similar wording close together.

One direction worth investigating is allowing representations to keep adapting to new feedback. But adaptation creates a difficulty of its own: if the representation space changes, can previously written states still be understood? We need not only to learn how to write, but also to maintain a usable relationship between old and new representations.

6.2 Associative Memory: Giving “Layered Notes” Rules for Reading and Writing

Associative memory uses cues to find content. A current question, for example, provides cues through which a model accesses relevant information from prior experience. Rapidly changing weights or states allow new cues and content to become associated inside the model; at read time, the current cue activates relevant information.

Simple superposition creates interference. Writing based on prediction error can instead correct an existing mapping, rather than merely add another trace. Research of this kind takes “walls” and “layered notes” from metaphor to update rules that can be analyzed.

Reading and writing natural language also require cue alignment: the same experience may be queried in different words, while similar wording may contain opposite conditions. How addresses are formed, how variation in expression is tolerated, and how crucial differences are preserved are central questions when associative memory enters real interaction.

The next generation will study capacity, retrieval reliability, and expressions of uncertainty together, so that structures already written can be used accurately in relevant situations.

6.3 First Learn How to Read and Write, Then Keep Taking in Experience: Lessons from TTT

“First teach it how to attach, find, and understand notes; then keep giving it new ones” is an important research idea. It separates a general learning mechanism from the changes caused by each particular experience.

Larimar connects encoding, distributed memory, and decoding to study rapid knowledge updates. MemoryLLM investigates a self-updatable internal memory pool. For us, their most valuable insight is that the coordination of reading and writing can itself be learned. Developing a mechanism for absorbing new experience, then allowing experience during use to continually update model state, is a path worth exploring in depth.

The key is to design representation, writing, and reading together. We will evaluate this coordination under device-resource constraints, long-term updating, and data-locality requirements, measuring separately the cost of developing a learning mechanism and the cost of using it day to day.

Test-Time Training (TTT) brings learning updates into model use. After encountering a new input, the model adjusts trainable parameters according to a learning objective, then uses the updated model to perform the task. The original TTT research constructs self-supervised tasks from the input itself, updating the feature extractor without human-provided labels for the current

sample to adapt to inputs that differ from the original training data. Training here genuinely changes parameters; generating additional rounds of answers or spending longer on inference is a different way of allocating computation. Related discussion is also available in this TTT post.

TTT updates can be retained in different ways. In the standard version of the original work, adaptation begins from the same starting point for each input, and the update is discarded after prediction. The online version carries forward the parameters updated on the preceding input. This distinction directly connects to NI's research question: how can a change learned in the current task become understanding that remains usable and revisable next time? TTT provides a method for updating during use; test-time continual learning further examines retention, consolidation, and long-term accumulation of those changes.

TTT Layers bring this idea into the model's internal sequence-processing architecture. "Learning to (Learn at Test Time)" makes the hidden state itself a small trainable model: when new input arrives, self-supervised learning updates its weights, and the updated state is then used to process information. Outer-loop training learns representations, transformations involved in reading and writing, and other model parameters; inner-loop updates compress information from the current sequence into fast-changing weights.

This offers a concrete reference for "first learn how to attach notes, then keep receiving new ones": a relatively stable learning mechanism can drive rapid changes during use. The research organizes fast state around input sequences. Applying such mechanisms to NI also requires designing retention across sessions, controlling interference among experiences, and consolidating fast state into long-term capabilities. We will study these questions alongside the multi-timescale design discussed next.

For NI, TTT is attractive because it makes learning a mechanism available during operation. We can investigate how update signals arise from prediction errors, input structure, or action feedback, and which parts should adapt quickly versus which judgments need more evidence before changing. Self-supervised updating is one concrete route; developing and using a learning mechanism does not necessarily depend on reinforcement-learning rewards.

"Learning to learn" becomes three concrete questions: how the system uses new evidence to decide what to update, how it distinguishes anomalies from patterns, and when it should revisit or revise an understanding it has already formed.

6.4 Multiple Timescales: Fresh Experience and Stable Capabilities Need Not Change at the Same Rate

Something that has just happened often needs to become usable quickly. A judgment developed over time should not be overturned by a single chance observation. If all content uses the same write strength, retention period, and forgetting rate, the system will struggle to be both responsive and stable.

Multiple timescales are an important direction for our next-generation research. Fast state handles immediate changes; slower state retains structures supported by more evidence. Periodic integration then determines which experiences deserve to become more stable capabilities.

Titans combines memory that can be updated during operation, forgetting mechanisms, and different information pathways, offering a reference for this design. We focus on how those mechanisms divide their roles and on the trade-offs among extra computation, state capacity, and long-term retention.

Consolidation emerges through effective coordination across timescales: new experience enters fast state, valuable structures gradually stabilize, and prior understanding is reorganized in light of new evidence. How information is exchanged and which conditions are retained determine the practical effect of this coordination.

Returning to the earlier memory distinction, a model can retain the important context of a specific experience and also develop more general understanding across experiences. These functions and their update rates need to be designed separately: some specific episodes deserve long-term retention, while some general judgments need rapid correction. We will organize updating and consolidation according to the role of the content and the strength of the evidence.

6.5 Writing, Forgetting, and Recall All Involve Trade-offs

Surprising experiences deserve attention: when outcomes differ greatly from predictions, prior understanding may need to change. An update mechanism must also distinguish meaningful new evidence from random noise, so that limited learning resources go toward changes that matter.

We will study novelty, reliability, task value, and long-term impact together. Frequent events can matter, but so can rare exceptions with serious consequences. Recently acquired understanding may be more current, yet understanding accumulated over time should not be erased casually.

An update gate determines how much to write, a forget gate determines what to retain or decay, and a reading mechanism determines what to use now. These controls need to relate to the importance, reliability, and applicability of the content, coordinating writing, retention, and access

within finite capacity.

Recall also requires selection. The value of continual learning is not in reviewing an entire lifetime before every answer, but in accessing useful structure for the present situation and recognizing what needs to be added when that structure is insufficient.

6.6 A Philosophical Lesson from Chips: How Can General Mechanisms Handle Future Tasks?

A chip does not need advance knowledge of every future program. As long as a program meets its instruction and execution requirements, the chip can compute using its existing mechanisms. This suggests a principle: **stable underlying mechanisms can support content and tasks that arrive later.**

For continual learning, we cannot prepare all of a user's future experiences in advance. We seek to understand whether a model can acquire a mechanism for absorbing new experiences without a separate set of semantic rules being written for every new domain, correction, and user.

Chips generally execute externally supplied programs. Agents must also discover patterns in incomplete, noisy feedback and even judge whether the questions they pose are appropriate. This gives a learning mechanism a further responsibility: it must both accept new content and assess the reliability of new understanding.

Our central conclusion from this analogy is that **stable underlying mechanisms can support an expanding range of knowledge and capabilities.** We are looking for learning mechanisms that accept later experiences, test new understanding, and keep developing capabilities.

6.7 Lessons from Heterogeneous Integration: Division of Roles and Interconnection, Not Just More Layers

Three-dimensional chip integration combines components with different functions and fabrication processes through close interconnection. The division of roles and approach to interconnection embodied in TSMC SoIC also offer a useful perspective on coordination inside a model.

Through this engineering analogy, we consider three questions: could different kinds of state use different representations? How should mechanisms with different update rates exchange useful information? Do the connections added to increase capacity bring acceptable reading and communication costs?

Capacity need not come only from a deeper backbone. It could also come from new modules, subspaces, or different forms of organization. Additional capacity becomes a practical capability only when learning mechanisms can write to it and inference can use it.

For us, the most valuable insight is to consider trade-offs at the system level: not only how much can be held, but how it can be accessed, updated, and coordinated—and what those capabilities cost.

7. What Are the Hard Problems We Need to Solve?

7.1 Compression Must Serve Tasks Not Yet Encountered

A representation designed only for the current question may discard a crucial cue for the next one. We need to balance the breadth of retained content, the depth of its organization, and resource cost, so that representations formed now can support a broader range of future questions.

Next-generation NI therefore faces not a single compression ratio, but a trade-off curve: within an explicit resource budget, how much useful structure can be retained, how broad a range of future tasks can it support, and where do omissions occur?

“Important” is consequently not a permanent label. An apparently minor experience may take on new meaning when the environment changes. Preserving the possibility of reinterpreting the past is a deeper problem than choosing a summary length.

7.2 How Does a Change in State Become a Change in Capability?

The hardest part may not be changing a matrix, but getting a model to use that change correctly.

Personalized expression, accurate recall, task efficiency, and judgment under unfamiliar conditions are different dimensions of improvement. Continual learning research must identify which changes come from patterns in experience, which depend on retaining particular content, and whether those changes can continue to accumulate.

We will compare behavior with and without learning updates while controlling context, tools, and computational budgets. Attributing contributions to specific mechanisms is how we determine which representations, update methods, or reading mechanisms to strengthen next.

7.3 From Correlation to Reliable Judgments About Action Consequences

Moving from associations among events to judgments about action consequences requires identifying environmental conditions, the timing of feedback, and potential contributing factors. The same action may have different results in different situations, and learning should preserve this conditionality.

The agent must therefore address attribution: after a success or failure, which step or condition should it revise its understanding of? Generalizing too early can produce false rules; treating every exception as a new rule can leave it unable to act coherently.

World modeling can contribute to this process: predict the consequences of an action, compare them with real feedback, and revise understanding through new observations and attempts. We see this testable relationship between prediction and action as a key step in turning understanding into capability.

7.4 Balancing New Experience with Retention of Existing Capabilities

Plasticity allows a model to accept new information; stability allows it to retain understanding that remains valid. Excessive stability rejects genuine change, while excessive plasticity can let a single anomalous experience lead the model astray.

We advocate learning mechanisms that preserve uncertainty, distinguish conditions of applicability, allow provisional explanations to coexist, and decide how to integrate them as more evidence arrives. This allows a model to accept change while resisting incidental noise.

Forgetting manages retention and decay, correction revises existing understanding, and deletion implements the user's decisions about what to keep. Each requires clear semantics, and all three belong in the learning-state lifecycle.

7.5 On-Device Constraints Must Be Part of the Algorithm

A learning-state file mounted for inference may be small yet expensive to update. A mechanism that reads quickly may require substantial auxiliary state. We will include the costs of encoding, updating, temporary storage, and restoration together in algorithm design.

We need to coordinate resource use between everyday inference and learning: lightweight updates should respond promptly, heavier reorganization should be scheduled appropriately, and ongoing interaction should remain usable. When computation happens, how much state is retained, and what updates achieve should be designed alongside the learning objective.

Differences in device memory, power use, and usage patterns lead to different algorithmic trade-offs. Our existing runtime work makes it possible to measure and compare these questions in real environments, grounding algorithm design in actual operating conditions.

7.6 Across Devices and Heterogeneous Environments: What Exactly Is Being Transferred?

We want users to carry the results of learning with them across devices. Achieving that requires addressing three kinds of transfer.

What is transferred	The question that needs to be answered
Learning mechanism	Can the same principle work with different computation, memory, and operating environments?
Learning state	Can existing state be restored under compatible conditions, and how should it be converted when representations differ?
Acquired capabilities	When sensing modalities, tools, and action spaces change, which parts of prior understanding remain useful?

We already have an implementation foundation for transferring and restoring compatible state, providing a practical path to learning continuity under compatible configurations of the same model.

Source: AtomGradient internal research.

The next step is to further investigate the connections among different model representations, sensing modalities, and action spaces.

“The model is the product” gives transfer a clear research objective: once a learning artifact enters a compatible device, it should support the use of prior understanding and continued learning from new experience. We will separately examine whether capabilities are retained, whether updating continues, and which states must be carried to support both. What the model needs from past experience should increasingly be embodied in its own acquired understanding.

Internal state depends on how a model represents information. Layer structures, numerical coordinates, and action semantics can differ across models, so transfer must address representation alignment and capability reuse. Applying prior planning experience to another kind of device, for example, requires connecting it to that device's sensing and execution methods.

The core of heterogeneous transfer is identifying which understanding can be shared and which details must adapt to a new platform. What we seek to carry forward is understanding and capability that can keep growing in a new environment.

8. Comparisons and Lessons from AgentOdyssey

8.1 What Observable Comparisons Does It Provide?

AgentOdyssey is a benchmark that uses procedurally generated game environments to observe how agents learn through extended interaction. Agents encounter new situations, choose actions, experience consequences, and then tackle problems that depend on earlier experience. Its value is in making the learning process and its effects observable.

It lets us ask more concrete questions: what environmental knowledge did the agent acquire? Did it retain important experiences? Did exploration yield valuable information? Did these changes help it advance the task?

The official evaluation includes task performance, world-knowledge questions before and after interaction, episodic memory, and exploration. These dimensions are better suited to analyzing the effects of experience than checking only whether the agent ultimately completes the game.

We will examine these dimensions together: what does recall contribute to judgment, what new understanding comes from exploration, and how does environmental knowledge change action? This lets us see the complete progression from retaining experience to using it.

8.2 Trade-offs with External Retrieval, Accumulated Context, and Other Approaches

A broader range of permitted methods gives us a rich set of comparisons. We are interested in what each approach solves, what it costs, and what it contributes to continual learning research.

Approach	What it provides well	What it helps us study
Continuing to add history to the context	Direct access to more experiences within the model's current computation	Compare direct use of history with internal accumulation in terms of effectiveness, latency, and resource growth
Storing material externally and retrieving it as needed	Inspectable, updatable original content and its sources	Compare the value of precise retrieval with that of internalized patterns, rather than only checking whether answers match
Organizing history into textual summaries	Conveying important prior information in less text	Examine what linguistic summarization preserves and how repeated summarization introduces distortion
Changing usable internal model state	Letting experience alter the conditions or mechanisms of subsequent computation	Examine whether changes persist, are used correctly, and incur acceptable updating and restoration costs

These approaches can be combined. NI focuses on internalizing experience and continual learning, while precise facts can be accessed through tools from authoritative sources. Through comparison, we identify the strengths, resource costs, and applicable conditions of different mechanisms before deciding how to organize them.

Comparisons can use the same starting point and similar resource budgets to identify a mechanism's contribution. They can also report performance across different budgets to assess practical value. Together, these perspectives support evidence-based trade-offs.

8.3 Applying These References to NI's Long-Term Learning Research

Drawing on AgentOdyssey's interaction tasks and evaluation dimensions, our experiments will examine adaptation within an environment, long-term retention, and accumulation across environments separately, then test these questions on real devices.

For example, resetting state between games is suited to observing how new understanding develops within one environment. Retaining state across environments helps investigate retention, transfer, and interference involving earlier experience. The protocol should match the learning question we actually want to answer.

AgentOdyssey's task organization and comparative methods help us analyze what knowledge an experience brings and how that knowledge changes later action. For NI, these references ultimately serve long-term learning on real devices: enabling models to keep building understanding and refining judgment and action through everyday use.

9. How Do We Establish That Continual Learning Has Taken Place?

9.1 Look Beyond the Outcome to Where the Change Came From

The next stage of experiments will address explicit learning questions, using corresponding observations and comparisons to identify the contribution of each mechanism.

Question	Evidence to examine
Has experience entered usable state?	Whether the specified learning state supports later tasks without supplying the same history again
Do updates actually have an effect?	Compare enabled and disabled updates while controlling additional context, tools, and computational budgets
Does the result go beyond repeating the original question?	Test new wording, combinations of conditions, and independent tasks, rather than simply repeating old questions

Question	Evidence to examine
Are capabilities retained and revised?	Add experience continually, checking prior capabilities, conflicting evidence, and unrelated tasks
Has learning continuity been established?	After restoration, check not only whether old state can be read, but also whether new experience can be accepted and updates remain effective
Does the mechanism have practical operating value?	Record the full state, update costs, inference costs, and operating reliability together

Cache reuse, recall, generalization from experience, and capability integration can coexist in one system. We will measure their contributions separately and observe how they support continued growth together.

Validation will also connect different levels: mathematical analysis helps explain capacity and interference; natural-language tasks test representation and reading; environmental interaction tests changes in action; and device experiments test the feasibility of long-term operation.

To study whether the model can carry its learning independently and across devices, we will also test whether, after restoring the complete learning artifact without supplying the original experiences again, it can recall relevant episodes, use existing knowledge, and keep revising its understanding with new feedback. This connects the states being carried with the capabilities being continued.

9.2 Let Scientific Questions Determine the Experimental Sequence

One reasonable sequence is to first establish a minimal working loop of representation, updating, and reading, then examine retention and revision across successive experiences, before moving to more complex actions, transfer, and device conditions.

This order makes problems easier to locate. If reading fails, we can first examine representation and cue alignment. If performance deteriorates over time, we can investigate capacity, interference, and update strategies. After state restoration, we can test whether new experiences continue to be absorbed correctly.

World-model evaluation likewise connects prediction and action: it measures both prediction accuracy and what those predictions contribute to decisions. Our aim is to make models better able to face the future.

9.3 Separate Learning Experiences from Independent Tests

Episodic-memory evaluation asks whether experiences have been retained; generalization evaluation asks whether acquired understanding can be used on unseen tasks. We report these capabilities separately so that each result has a clear meaning.

Answers to independent tests, hidden environmental information, and scoring criteria should be kept separate from the learning process. Tasks used to select a method should also be distinct from those used for final evaluation. This design allows results to reflect the actual contribution of learning.

Semantic targets and evaluation criteria for real data need to come from an independent, traceable evaluation process. We test different methods under consistent conditions and report both successes and failures faithfully.

Real records help us study the distribution of everyday information, while interactive environments add actions and feedback. We will cover different users and scenarios and progressively examine the range of applicability. Keeping data local and obtaining user authorization remain prerequisites throughout.

9.4 Resource Accounting Must Cover Both Use and Continued Learning

Resource evaluation must distinguish at least three things: what the learning process must retain, what must be loaded for inference, and what is needed for the next update.

It must also count input context, temporary memory, encoding computation, and state-restoration costs. Dependencies such as external indexes and auxiliary update state belong in a method's full resource accounting as well.

We evaluate approaches by the relationship between capabilities and their full costs, judging whether a mechanism can progress from experiment to long-term use.

10. Our Research Choices and Open Trade-offs

Our direction is clear: NI has always pursued continual learning; intelligence goes to the data; models and their learning states carry what has been learned; and growth should continue on real devices in changing environments.

The implementation remains open. We select methods according to how well representation, updating, and consolidation work in practice, and assess research contributions by continued capability development and long-term operating value.

10.1 Which Changes Should Become Part of the Model?

Temporary conditions of the current task, observations awaiting confirmation, and well-established patterns should be handled differently. Allowing important understanding to gradually become established is a central trade-off in continual learning under finite resources.

A learning mechanism should make clear which changes are retained, on what basis, and how they are revised when new evidence arrives. Selecting structures worth accumulating over time is itself a necessary capability of continual learning.

10.2 How Should Capacity Growth and Consolidation Be Coordinated?

Fixed budgets, budgets tailored to device capabilities, and moderate capacity expansion each have costs. We will study capacity management together with experience consolidation, making better use of existing state while evaluating the benefits and costs of measured expansion.

Rapid absorption, periodic integration, and longer-term capability retention need coordinated rhythms. The next generation will investigate information exchange and scheduling among these processes, so that everyday updating and long-term accumulation support one another.

10.3 How Can Autonomous Learning Coexist with User Control?

Environmental feedback helps a model judge what deserves attention. Users determine the scope of learning and the boundaries for state retention and transfer. Initiative in learning and human control need to be realized within the same design.

An agent should actively observe, update, and review within clearly authorized boundaries, while users can understand, undo, and restore the relevant changes. Control extends throughout the learning process.

10.4 Different Devices Can Take Different Roles

Devices differ in computing power and usage patterns. Everyday adaptation and heavier reorganization need not happen at the same time or in the same way.

This division of work will depend on data remaining local, compatibility, and user authorization, with computation allocated according to each device's resources and capabilities. We will validate it under specific device conditions so that learning continuity matches what can actually run.

We hold to clear goals and revisable methods: knowing why we do the work, while allowing evidence to change how we do it.

11. Conclusion: Models That Keep Developing Capabilities Through Experience

From its first generation to the next, NI has always centered on continual learning. Paper notes and walls help us think about how experience is organized; compression and forgetting help us consider what should be retained; games and world models help us examine how actions, consequences, and understanding affect one another; and the general mechanisms of chips inspire us to ask how to handle tasks that have not yet appeared.

We have also done practical work on these questions. The technology stack developed around NI connects model optimization, inference runtimes, learning mechanisms, tool integration, and cross-device state management, giving continual learning an operational foundation on devices. On-device incremental learning already connects new records, updates to personal characteristics, and subsequent inference. The next generation will build on these foundations to investigate finer-grained updates, experience consolidation, ongoing correction, and learning continuity across devices.

We are working toward a model that continually develops understanding, refines judgment, improves action, and revises itself in the face of new evidence. Memory participates in this process; the ongoing development of capability remains its purpose throughout.

The device is the agent; the model is its intelligent core, capable of growth and transfer; and NI enables that growth to continue. “The model is the product” expresses our aspiration for this accumulation: the understanding and capabilities formed through long-term interaction should travel with the model to new compatible devices and continue developing through new experiences.

When intelligence goes to the data, growth should happen where experience takes place.

We test this path with a simple question: without repeated reminders or being given the same history again, is the model genuinely better able to handle the next situation because of what it has experienced before?

Appendix A: Quick Glossary

These explanations support the reading of this paper; no prior familiarity with every research field is assumed.

Term	Meaning in this paper
Neural Imprint (NI)	A continual learning algorithm and algorithmic architecture developed by AtomGradient, with a supporting technology stack that enables models to run, learn, and keep growing on devices
Device agent	A computing-capable device that brings together a model, sensing and action interfaces, and learning state to organize observation, action, feedback, and learning
Continual learning	Acquiring, retaining, and revising knowledge and capabilities through successive experiences while addressing environmental change and retention of earlier capabilities
Incremental learning	Absorbing new information on the basis of prior learning, without assuming that each update starts from scratch
Test-Time Continual Learning (TTCL)	Continued adaptation and capability consolidation during deployment and use, rather than training only before delivery
Test-Time Training (TTT)	Updating trainable parameters during use according to a learning objective; updates may serve the current input or be retained afterward, depending on the design
TTT Layers	Sequence-processing layers whose hidden state is a small trainable model, compressing information from the current sequence through learning updates during use
The model is the product	Product value accumulates as the model learns; acquired understanding and capabilities are carried by the complete model and its internal learning states, with transfer and continued growth as goals
Model weights (parameters)	Numerical values used in model computation; training can adjust them, while inference uses them to process input
Inference runtime	Software that arranges model computation, memory use, and state management, enabling a model to run on a particular device
Internal state	Numerical information that can be read or updated during model computation; this paper focuses on the parts that can carry learning outcomes and continue to participate in computation
Activations and representations	Internal numerical values formed as a model processes input; representations help organize the content and relationships within that input

Term	Meaning in this paper
Latent representation	An internal form used to represent content and relationships; the information it retains determines what can later be read or generated
RPP	A mechanism for analyzing personally relevant structure in model activations, involving residualization, principal component analysis, and stability checks
DSR (Dual-Sparse Retention)	Dual-sparse token retention: a mechanism in our runtime that manages historical cache retention using attention scores, recent content, and layer-specific budgets
FrogJump	An inference execution mechanism that skips selected computation layers according to a plan, under supported models and operating modes
Episodic memory	Retention and recall of specific experiences and their context, including conditions, actions, and consequences; discussed functionally here without presupposing a separate module
Semantic memory	Knowledge of facts, concepts, and patterns that can be used without replaying the experience in which it was first acquired
Factual memory and exact factual recall	The former broadly refers to retaining factual content, which can occur in episodic or semantic memory; the latter specifically refers here to accurately answering questions about the details of original records
Adaptive reasoning	Addressing new problems by combining prior understanding with current conditions, and revising judgments based on new evidence; memory supplies material for this process
World model	A model that internally predicts environmental states and action consequences, supporting internal simulation and correction through real feedback
Associative memory and fast weights	Mechanisms that use cues to establish and read associations, and allow internal mappings to change rapidly with experience
Consolidation	Organizing new and old experiences into relatively stable, reusable understanding or capabilities, without restricting them to one particular form of storage
Plasticity and stability	The ability to accept new information and the ability to retain understanding that remains valid; the two need to be coordinated
Activation Steering	Adjusting internal activations during inference to influence subsequent computation, without requiring changes to model weights
Directional Steering	Our specific path for deriving directions from corrections and using them to influence subsequent computation; an implementation of activation intervention
Generalization and transfer	Generalization concerns effectiveness on unseen inputs; transfer further concerns reuse across different tasks, environments, or platforms

Term	Meaning in this paper
Heterogeneity	Differences in chips, operating environments, model representations, sensing modalities, or action spaces; discussion should specify which kind of difference is meant
Token	A unit of sequence processing in a model; it does not necessarily correspond to one Chinese character or one word
Time to first token	The wait from the start of the relevant request to the first output token, with the precise timing boundaries defined by the experiment

How to cite

AtomGradient. Next-Generation NI: A Vision, Philosophy, and Technical Path for Continual Learning (Whitepaper v2). 2026-09-17. https://atomgradient.github.io/whitepapers/neural-imprint/en/

```
@techreport{atomgradient2026ni_en,
  title  = {Next-Generation NI: A Vision, Philosophy, and Technical Path for
Continual Learning},
  author = {{AtomGradient}},
  year   = {2026},
  month  = {09},
  type   = {Whitepaper v2},
  url    = {https://atomgradient.github.io/whitepapers/neural-imprint/en/}
}
```

Read online: https://atomgradient.github.io/whitepapers/neural-imprint/en/ · Changelog: github.com/AtomGradient/whitepapers · Website: atomgradient.com