Differentiable Exact Learning

Hristo Papazov* **Francesco D'Angelo** **Nicolas Flammarion**

Department of Computer Science, EPFL

Abstract

We introduce a framework for differentiable exact learning that couples an unbounded external environment to a state-tracking neural controller, which we train on policy-trajectory observations (PTOs): expert behavior traces that expose observations and actions while hiding internal states. In addition to the learning framework, we introduce Differentiable Finite-State Transducers (DFSTs) which use multilinear updates, support log-parallel training, and emulate arbitrary finite-state controllers, yielding computational universality through environmental interaction. From tiny training datasets, compact DFSTs achieve extreme length generalization on arithmetic and universal-simulation tasks. We replicate the success of DFSTs on RNNs and GRUs, thereby supporting a broader learning recipe: **State-Tracking Controller + Unbounded External Environment + PTOs = Extreme Length Generalization**. Beyond length generalization, our metric-extraction protocol recovers interpretable discrete controllers from trained DFSTs, which we prove equivalent to the target arithmetic and universal-simulation algorithms. Furthermore, for our Softmax DFST models, simply taking argmax over the transition probabilities yields discrete controllers which provably exactly perform universal Turing-machine simulation and binary arithmetic. Proving exact learning for the neural controllers requires assumptions, which we empirically support. We make our code, training recipes, model weights, and proof checkers publicly available.

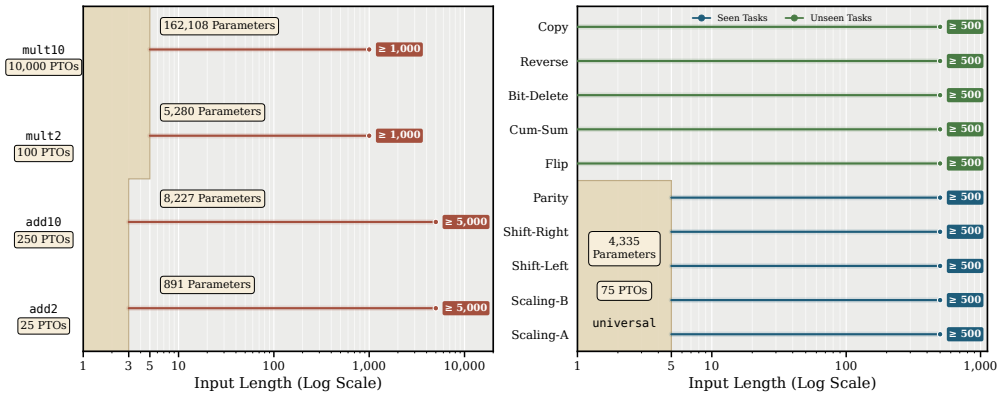


Figure 1: **Extreme Length Generalization from Small PTO Datasets and Compact Controllers.** Length generalization by DFST models trained from scratch on arithmetic and universal-simulation tasks. Shading marks the length of inputs seen during training. Our models see computation traces of inputs with ≤ 3 digits for addition and ≤ 5 digits for multiplication. On the universal task, DFSTs see simulation traces of 5 Turing machines on inputs of up-to 5 symbols and generalize to 5 unseen Turing machines. Endpoints mark successful length-generalization reaching a pre-defined testing cap.

*Corresponding author: `hristo.papazov@epfl.ch`

40th Conference on Neural Information Processing Systems (NeurIPS 2026).

1 Introduction

For decades, researchers have sought to make neural networks learn algorithmic procedures from data, arguing that genuine machine intelligence requires exact recovery of underlying rules rather than matching surface-level statistics (Solomonoff et al., 2006; György et al., 2025).

Yet arithmetic, the canonical testbed for algorithmic learning, continues to expose a persistent gap. Transformers and RNNs trained on input-output data generalize only marginally to longer inputs (Delétang et al., 2022). Even an extensive arsenal of scratchpad protocols, positional-encoding schemes, and input-formatting tricks brings Transformer length generalization[2] to 2–7$\times$ the training operand length, with exact-match accuracy swiftly decaying as inputs lengthen (Hou et al., 2025; Y. Zhou et al., 2024; Cho et al., 2024a). The Neural GPU (Kaiser and Sutskever, 2015) stands out by reaching $100\times$ length generalization on random binary arithmetic instances, although later diagnostics expose long-carry and symmetric-input failures (Price et al., 2016) (see Table 1). Such finite-test success leaves exactness on arbitrarily long inputs unresolved.

A relevant exception comes from the recursive Neural Programmer-Interpreter (NPI) (Reed and De Freitas, 2015; Cai et al., 2017), which provably recovers decimal addition and several sorting algorithms. However, the training uses a stronger form of supervision through recursive call/return annotations that goes beyond merely observing expert actions in an external environment.

In contrast, real learners routinely succeed under exactly this restriction. Indeed, children acquire complex procedural skills by observing adults perform goal-directed actions, with no access to the adults' internal neural processes. From such behavior traces alone, infants infer the underlying procedural structure and reproduce the target behavior through practice (Meltzoff and Moore, 1995).

Motivated by the success of natural learners, this paper entertains the idea of gradient-based exact algorithmic learning from behavior observations and provides a simple recipe for achieving extreme length generalization with compact controllers trained from scratch on small datasets (see Figure 1). To our knowledge, this work provides the first differentiable exact recovery of a fixed *universal interpreter* with correctness proofs covering arbitrary simulations of Turing machines with unbounded input lengths and unbounded execution horizons. We similarly achieve exact differentiable recovery on addition and multiplication tasks. Our training withholds expert control states and the learned interpreter requires no retraining when exchanging input programs. We just need two ingredients: sufficiently informative observations and a sufficiently expressive learner.

Sufficiently Informative Observations. The choice of training data determines the possibility of exact learning. Gold (1967) proved that no algorithm can identify the class of recursive functions in the limit from input–output examples alone (Gold, 1967, Theorem I.5). Moreover, the impossibility persists in the active-learning setting (Gold, 1967, Theorem I.3). Hence, supervised learning from $(z, f(z))$ pairs and reinforcement learning rewarded only for correct output without a helpful curriculum cannot identify every recursive function in general. To navigate between this general identification barrier and the additional (strong and unrealistic) program-structure supervision of NPIs, we adopt policy-trajectory observations (PTOs) as our training signal. PTOs record how the environment evolves under the expert's policy without exposing the expert's internal state transitions.[3] This middle ground carries two advantages: (i) PTOs avoid Gold's impossibility barrier (Papazov and Flammarion, 2025, Theorem 16); (ii) PTOs admit a natural interpretation as behavior observations – the learner only observes how the world changes under the expert's actions.

Sufficiently Expressive Learner. Since we target exact learning of arbitrary algorithms, the learner must belong to a Turing-complete model class. Prior work establishes Turing-completeness for Transformers and RNNs through assumptions that preclude effective optimization: unbounded arithmetic precision (Siegelmann and Sontag, 1992; Pérez et al., 2019), unbounded context windows (Dehghani et al., 2018; Bhattamishra et al., 2020; Merrill and Sabharwal, 2023a), or non-differentiable environmental interaction (Chung and Siegelmann, 2021; Schuurmans et al., 2024). We propose a simpler workaround that preserves finite precision and fixed-size contexts, allowing for effective gradient-based optimization. The key idea couples a state-tracking[4] controller – such as an RNN, LSTM, or GRU – to a dynamically resizing external environment in which the controller can move, read, and write. Because the environment grows as needed, computation no longer bottlenecks on the

[2]Defined in the referenced papers as achieving above 95% exact-match accuracy (EMA).

[3]Recovering an algorithm with PTOs mirrors more realistic learning-from-demonstration settings (Argall et al., 2009).

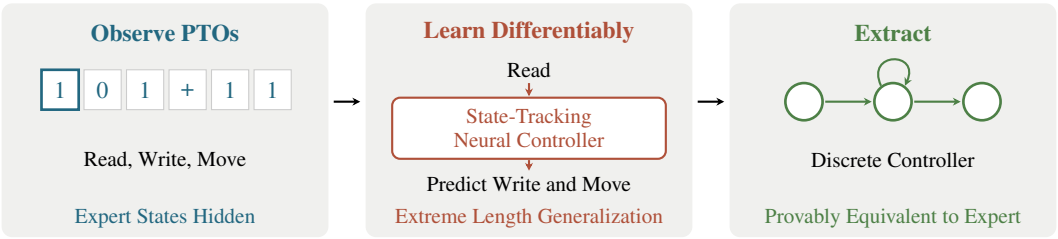
[4]We define state-tracking as capable of simulating arbitrary finite-state automata given sufficient model capacity.

controller's fixed internal memory. Zaremba and Sutskever (2015) explored a related architecture pairing an LSTM controller with external tapes, but trained via reinforcement learning on input–output rewards alone, which Gold's theorem renders insufficient for exact learning in general. We hope to revive and strengthen this architectural idea through the richer PTO supervision.

Thus, our learning recipe distills into a simple equation:

State-Tracking Controller + Unbounded External Environment + PTOs = Extreme Length Generalization.

In fact, we go a step further and introduce a simple extraction procedure, which recovers discrete controllers from our trained neural models. We prove that the extracted controllers exactly follow the target behavior and hence certify the following method for differentiable exact learning of algorithms:



Main Contributions.

1. **A Conceptual Framework for Differentiable Exact Learning.** We revive the idea of achieving Turing-complete differentiable computation through a state-tracking neural controller coupled to an unbounded external environment. Training on a small number of PTOs yields extreme length generalization. Subsequent extraction establishes exact learning for the recovered discrete controllers across the studied tasks and for the neural controllers subject to stability assumptions.
2. **Differentiable Finite-State Transducers (DFSTs).** We introduce a (i) minimalist, (ii) multilinear, (iii) differentiable, (iv) state-tracking, and (v) parallelizable model family whose members serve as controllers in our experiments. DFSTs use no nonlinear activations, admit log-parallel training via prefix scans (Blelloch, 1990), and offer a transparent structure for studying the loss landscape of algorithmic learning and the useful implicit bias of gradient descent.
3. **Extreme Generalization from Small PTO Datasets.** We train on small PTO datasets (25 traces for `add2`, 100 for `mult2`, 250 for `add10`, and 10000 for `mult10`) and reach robust testing caps of 5000 digits for addition and 1000 digits for multiplication. The training operands contain at most 3 and 5 digits, respectively. We replicate the extreme length generalization on binary arithmetic using RNN and GRU controllers thereby supporting our generalization recipe.
4. **Exact Learning of a Universal Controller.** A 4335-parameter DFST trained from scratch on just 75 PTOs from 5 program simulations over inputs of at most 5 symbols reaches the 500-symbol evaluation cap across 10 tasks, including 5 held-out programs. From trained DFST models, our extraction procedures recover controllers provably equivalent to the expert universal interpreter and capable of simulating arbitrary programs on any input lengths and execution horizons.
5. **Extraction and Verification of Exact Learning.** A lightweight extraction procedure applied to trained DFSTs recovers compact FSTs. Eight proofs establish exact expert-policy equivalence across binary and decimal arithmetic and universal simulation. Additional empirically verified assumptions prove the equivalence of the trained neural controllers to the experts.

Remark. An earlier workshop version introduced DFSTs and reported considerable arithmetic length generalization (Papazov et al., 2025). The present work simplifies training; pushes length generalization to the extreme; and adds learned universal simulation, Softmax DFSTs, recurrent-controller comparisons, and exact-equivalence certificates for extracted controllers.

Paper Organization. Section 2 formalizes PTO-learning, DFSTs, and the studied algorithmic tasks. Section 3 expands on our arithmetic and universal experiments, with a short passage on the implicit bias of gradient descent. Section 4 explains FST extraction and verification of correctness. We refer the reader to Section A for a thorough discussion of related work.

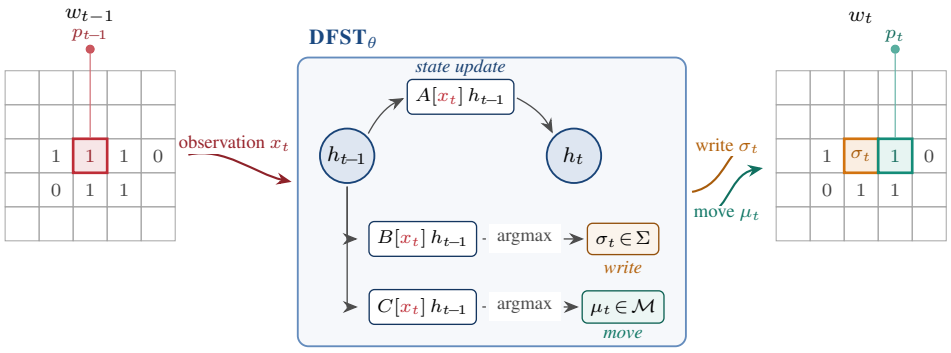


Figure 2: **One Step of the DFST Agent.** *Left:* the grid w_{t-1} before the action; the agent at position p_{t-1} reads the symbol in the red cell as observation x_t. *Center:* the DFST$_\theta$ controller updates the hidden state via $h_t = A[x_t]\,h_{t-1}$ and produces the discrete outputs $\sigma_t = \arg\max(B[x_t]\,h_{t-1}) \in \Sigma$ (write) and $\mu_t = \arg\max(C[x_t]\,h_{t-1}) \in \mathcal{M}$ (move). *Right:* the resulting grid w_t; the orange cell shows the symbol σ_t written at p_{t-1}, and the agent has shifted to p_t in the teal cell. We provide an interactive viewer that animates expert and trained neural controllers.

2 Preliminaries

As discussed in the introduction, our recipe for exact learning couples a state-tracking controller to an unbounded external environment.

2.1 Policy-Trajectory Observations

We model the external environment as a symbolic grid.

Definition 2.1 (Symbolic Grid)**.** Given a finite alphabet Σ containing the empty symbol λ, a *grid configuration* $w : \mathbb{Z}^2 \to \Sigma$ assigns a symbol $w(p) \in \Sigma$ to each cell $p \in \mathbb{Z}^2$ such that only finitely many cells differ from λ. The symbolic grid $\mathcal{G} = \mathcal{G}(\mathbb{Z}^2, \Sigma)$ denotes the set of all such configurations.

The finiteness requirement captures the idea that the environment grows as needed but never stores an infinite amount of information at any point in time. We now formalize the agents that inhabit the grid.

Definition 2.2 (Grid Agent)**.** A *grid agent* over the symbolic grid $\mathcal{G}(\mathbb{Z}^2, \Sigma)$ stands for a finite-state transducer $T = (Q, \Sigma, \Gamma, \delta, \gamma, q_0, q_f)$, where Q denotes a finite set of states, Σ serves as both the read alphabet and the write alphabet, $\Gamma = \Sigma \times \mathcal{M}$ with $\mathcal{M} = \{\mathtt{L}, \mathtt{R}, \mathtt{U}, \mathtt{D}, \mathtt{S}\}$ forms the output alphabet pairing a grid symbol with a motion direction, $\delta : Q \times \Sigma \to Q$ specifies the state-transition function, $\gamma : Q \times \Sigma \to \Gamma$ specifies the output function, $q_0 \in Q$ denotes the initial state, and $q_f \in Q$ denotes the halting state. The agent always starts at $p_0 = (0, 0)$.

The agent occupies a single cell $p_t \in \mathbb{Z}^2$ at each time step t, reads the symbol $x_{t+1} = w_t(p_t)$, produces an output $\gamma(q_t, x_{t+1}) = (s_{t+1}, m_{t+1})$ that writes s_{t+1} at position p_t and shifts the agent's position by m_{t+1}, and transitions to state $\delta(q_t, x_{t+1})$. We adopt the convention that every transition containing the stop motion $\mathtt{S}$ leads to the halting state q_f, and all transitions from q_f satisfy $\delta(q_f, x) = q_f$ and $\gamma(q_f, x) = (x, \mathtt{S})$ for every $x \in \Sigma$, rendering q_f a dead state that leaves the grid unchanged.

Note that a grid agent perceives only one cell at a time. Future work could explore richer perception mechanisms encoding the contents of multiple cells and agents interacting with a larger motion vocabulary. Since standard Turing machines arise as a special case of grid agents restricted to a single row, the class $\mathcal{A}$ of all grid agents achieves computational universality when interacting with $\mathcal{G}$.

PTOs. At each time step t, an external learner observing agent T perceives only the written symbol $s_t \in \Sigma$, the motion $m_t \in \mathcal{M}$, and the observation x_t that triggered the action. Importantly, the internal state q_t remains hidden. The resulting sequence $(x_t, s_t, m_t)_{t=1}^{T}$ constitutes a *policy-trajectory observation* (PTO), matching the formalism of Papazov and Flammarion (2025). Crucially, Papazov and Flammarion (2025, Theorem 16) proved that PTOs enable exact identification of any computable function in the limit – a guarantee unachievable through input–output supervision alone (Gold, 1967).

Hardness. Each algorithmic task $\mathcal{P}$ comes with a set of valid inputs $S_{\mathcal{P}} \subseteq \Sigma^\star$. Because the motion token outputted at each step constrains the next observation, $S_{\mathcal{P}}$ may never cover all reachable transducer transitions, thereby making the learning problem fundamentally hard. Papazov and Flammarion (2025, Theorem 17) proved that no *polynomial-mass* characteristic set exists for this restricted-input regime, ruling out any algorithm that learns the policy of every grid agent from polynomially many polynomially long PTOs (see Section C.3 for details).

2.2 State-Tracking Differentiable Controllers

While Papazov and Flammarion (2025, Theorem 16) showed that a learning-by-enumeration strategy provably identifies any grid agent from PTOs, such an approach takes time and memory exponential in the size of the target expert agent. We instead pursue gradient-based learning by introducing a simple differentiable model that can serve as a grid-agent controller.

Definition 2.3 (DFST)**.** A *Differentiable Finite-State Transducer* (DFST) of dimension d over the symbolic grid $\mathcal{G}(\mathbb{Z}^2, \Sigma)$ with motion vocabulary $\mathcal{M}$ comprises three trainable weight tensors

$$A \in \mathbb{R}^{|\Sigma| \times d \times d}, \qquad B \in \mathbb{R}^{|\Sigma| \times |\Sigma| \times d}, \qquad C \in \mathbb{R}^{|\Sigma| \times |\mathcal{M}| \times d},$$

and a trainable initial hidden state $h_0 \in \mathbb{R}^d$. At each time step $t \geq 1$, the controller reads the one-hot encoding $x_t \in \mathbb{R}^{|\Sigma|}$ of the symbol at the current grid cell $w_{t-1}(p_{t-1})$ and computes

$$h_t = A[x_t]\, h_{t-1}, \qquad \hat{s}_t = B[x_t]\, h_{t-1}, \qquad \hat{m}_t = C[x_t]\, h_{t-1}.$$

A drives the hidden-state recurrence, B produces scores over symbol tokens, and C produces scores over motion tokens. At inference time, the controller emits $\sigma_t = \operatorname{argmax}(\hat{s}_t) \in \Sigma$ and $\mu_t = \operatorname{argmax}(\hat{m}_t) \in \mathcal{M}$, writing σ_t at position p_{t-1} and moving μ_t to p_t (see Figure 2).

The DFST recurrence and readouts use only input-indexed matrix-vector products, without nonlinear activations. This multilinear structure enables log-parallel training via prefix scans (Blelloch, 1990) and yields a compact parametrization. A d-dimensional DFST over an alphabet of size $k = |\Sigma|$ with $r = |\mathcal{M}|$ motion tokens carries $kd(k + d + r) + d$ trainable parameters, scaling as $O(d^2)$.

Definition 2.4 (Softmax DFST)**.** A Softmax DFST has the same recurrence and readout mechanism as a DFST, with trainable logits $\Theta^A, \Theta^B, \Theta^C, \theta^h$ parametrizing

$$P[x]_{:j} = \operatorname{softmax}(\Theta^P[x]_{:j}), \quad P \in \{A, B, C\}, \qquad h_0 = \operatorname{softmax}(\theta^h).$$

Each column sums to one. Thus, the hidden state remains in the probability simplex and the readouts give distributions over writes and motions. Execution uses the same argmax decoding as a DFST.

This parametrization makes extraction transparent: each column assigns probabilities to successors or actions, and concentration near a simplex vertex suggests a deterministic transition. The recurrence retains the associative matrix scan. Section 4 develops differentiable sharpening and the conditions for faithful extraction.

Despite the simplicity of these architectures, both families support state tracking:

Theorem 2.5 (DFST-Grid Universality)**.** *Any d-state grid agent admits exact emulation by a d-dimensional DFST at any finite arithmetic precision, and by a $(d+1)$-dimensional Softmax DFST with finite logits in exact real arithmetic. A d-dimensional Softmax DFST also suffices under rounded arithmetic whose softmax attains simplex vertices and whose recurrence preserves vertex operations.*

Consequently, both the DFST and Softmax DFST families achieve Turing-completeness on the symbolic grid under the respective arithmetic conditions. Section B gives the constructive proof and discusses non-uniqueness of DFST parametrizations and the relationship between hidden dimension and expert state count.

More broadly, any sequential model capable of emulating a finite-state transducer can represent the policy of a grid agent and thereby compute any computable function on the grid. This class includes RNNs, LSTMs, and GRUs (Alon et al., 1991; Indyk, 1995; Siegelmann, 1996; Weiss et al., 2018; Merrill, 2019; Svete and Cotterell, 2023) whereas standard Transformers and diagonal SSMs face state-tracking limitations as demonstrated by Merrill and Sabharwal (2023b); Merrill et al. (2024). We focus on DFSTs due to the multilinear structure and simplicity of the model class, which we hope will inspire theoretical analysis of the implicit bias of gradient descent toward generalizing solutions. The DFST, RNN, and GRU experiments in Section 3 support this broader state-tracking perspective.

2.3 Experimental Setup

We study grid agents for binary and decimal arithmetic (`add2`, `add10`, `mult2`, `mult10`) and a universal grid agent U that simulates Turing machines over the $\{\lambda, 0, 1\}$ alphabet. Table 2 summarizes expert state counts, alphabets, and runtimes for all implemented grid agents and Turing machines.

Arithmetic Tasks. Each arithmetic agent receives $z = a \circ b$, with $\circ \in \{+, \times\}$, written horizontally on an otherwise empty grid. The agents execute the schoolbook algorithms for addition and multiplication, halting when only the result remains on the grid.

Data Generation. Each arithmetic PTO dataset includes all R^2 pairs of single-digit operands, then samples distinct pairs with independent uniform operand lengths in $1, \ldots, q$ and digits in $0, \ldots, R-1$. Leading zeros remain part of the representation. Ordinary DFST datasets always use: `add2`: $q = 3$, $N = 25$; `add10`: $q = 3$, $N = 250$; `mult2`: $q = 5$, $N = 100$; `mult10`: $q = 5$, $N = 10$k.

Length-Generalization Metrics. Uniform Length Generalization (ULG) records the operand length returned by a capped search requiring perfect next-action exact-match accuracy on ten random pairs of exactly that length. Robust Length Generalization (RLG) additionally requires perfect accuracy on structured repdigit probes and cannot exceed ULG. Every RLG evaluation tests all R^2 repdigit pairs (a^m, b^m) with $a, b \in \{0, \ldots, R-1\}$. Every timestep must match both the expert write and motion. The Neural GPU motivates this distinction: strong generalization on random binary inputs (Kaiser and Sutskever, 2015) can coexist with failure on repdigit instances (Price et al., 2016). Section D.6 specifies the search, numerical execution modes, and limits of finite testing.

Learning a Universal Interpreter. The universal grid agent U uses 15 states and alphabet $\{0, 1, \lambda, X, Y, Z, \mathtt{L}, \mathtt{R}, \mathtt{S}\}$. Given an encoding $\langle M, z\rangle$ of a Turing machine M and input $z \in \{0,1\}^+$, U simulates M on z, halting with $M(z)$ in a designated grid region whenever M halts.

U uses 3 teleportation runes $\{X, Y, Z\}$ to mark and jump directly between grid locations, shortening simulation trajectories. The grid stores an encoding of the currently simulated Turing machine M together with the input. From PTOs of the universal expert, a differentiable controller learns how to read and execute this program, without access to the internal states or transition map of U. The public repository provides model weights and animated executions.

Data Generation and Evaluation. Universal datasets contain $N \in \{75, 150, 225\}$ PTOs covering 5 programs, with binary input lengths at most 5. After sampling a random string for each program, the rest of the dataset comes from random program–input pairs. Evaluation covers 10 tasks, including 5 held-out programs and scaling tasks that increase both input length and program size. Section F details the rune mechanics, program/input encoding, and training and evaluation tasks.

Controller Training. For each task, a manually designed expert supplies PTOs. We train controllers via next-action prediction with the MSE loss

$$\frac{1}{2T} \sum_{t=1}^{T} \left[\|\hat{s}_t - \mathbf{1}_{s_t}\|_2^2 + \|\hat{m}_t - \mathbf{1}_{m_t}\|_2^2 \right],$$

where $\mathbf{1}_{s_t}$ and $\mathbf{1}_{m_t}$ denote the one-hot encodings of the expert write and motion at time t. We train DFST controllers on all tasks, study Softmax DFSTs on binary arithmetic and universal simulation, and compare RNN and GRU controllers on binary arithmetic. Training always uses Adam (Kingma, 2014), batch size 32, and `float32` precision.

DFST Training. Unless otherwise specified, DFST hidden dimensions match the expert state counts. We always use a simple deterministic initialization. Transition matrices start at the identity, output matrices at zero, and the trainable initial state at $\mathbf{1}_d$. All ordinary DFSTs runs use an exponential warm-up and cool-down learning-rate scheduler(Sections D.1 and D.2). Section B.3 examines successful learning with smaller and larger hidden dimensions.

Softmax DFST Training. Softmax DFSTs apply the same initialization recipe to the logits before softmax normalization. The sweeps use one, two, or three times the expert state count and vary dataset size. Training uses 100k updates at constant learning rate 0.1 (and optional annealing towards approximately one-hot transition probabilities). Section D.4 gives the full training settings.

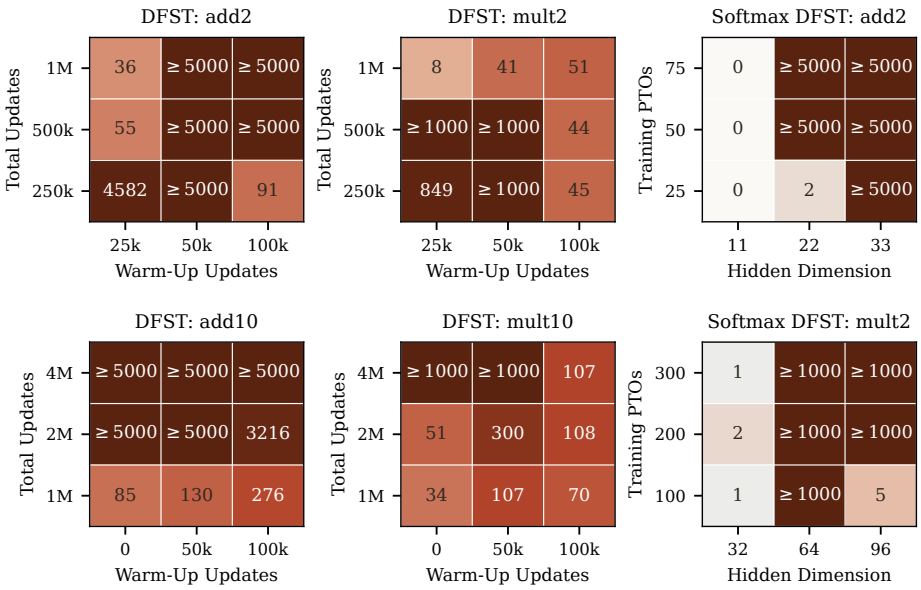


Figure 3: **Robust Generalization on Arithmetic from Differentiable Training.** The first two columns show ordinary DFST learning-rate schedule sweeps. The last column shows Softmax DFST dimension/data sweeps after 100k updates of constant learning-rate training. Cells report RLG in operand digits. Section E provides the paired ULG/RLG plots.

3 Experimental Results and Discussion

In this section, we discuss length-generalization results solely due to differentiable training.

3.1 Length Generalization on Arithmetic

Figure 3 compares arithmetic generalization before extraction. All DFST and Softmax DFST models train from scratch with the same simple deterministic initialization recipe.

Several DFST runs reach both the ULG and RLG testing caps: 5000 digits per operand for addition and 1000 for multiplication, in binary and decimal radix, after training on operands of at most 3 and 5 digits, respectively. Success varies across training schedules, and additional updates need not improve generalization.

After 100k constant learning-rate updates, many Softmax DFST configurations reach the robust binary-addition and multiplication caps. The Softmax DFST runs hold the advantage of providing interpretable transition probabilities while also supporting extreme generalization. However, Softmax DFST optimization proves more challenging, and we could not fit even small training sets on the decimal tasks. Section D.7 discusses these limitations.

Tens of Millions of Consecutive Correct Actions. Operand length understates the scale of this extrapolation. Adding two 5000-digit all-9 numbers requires 50 070 010 PTO steps; multiplying two 1000-digit all-9 numbers requires 15 012 991. Each step requires the correct write and motion, including the final stopping action. The corresponding binary all-1 probes require exactly the same numbers of steps (Section F.5). Successful decimal DFSTs reproduce the all-9 trajectories, while multiple binary DFST and Softmax DFST configurations reproduce the equally long all-1 trajectories. The binary controllers learn from only 25–75 addition PTOs or 100–300 multiplication PTOs on operands of at most 3 or 5 digits, respectively. Such training supports faithful execution over tens of millions of steps on these structured probes, far beyond the short training traces.

Small-Data Learning and Implicit Bias. These long executions raise a question beyond representational capacity: why does gradient-based training favor rules that continue working so far beyond the training lengths? Matching the training PTOs alone does not force such behavior. We construct spurious binary addition and multiplication controllers that reproduce every expert PTO within the

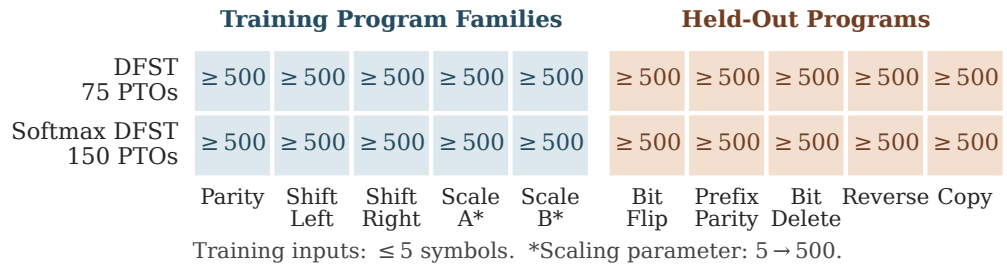


Figure 4: **Generalization on Universal Simulations from Differentiable Training.** Cells report successful tested input lengths under teacher-forced exact action matching. Both controllers reach the 500-symbol cap after training on inputs of at most 5 symbols. Figure 13 gives the full sweeps.

training-length bounds, yet stop prematurely on longer inputs. Thus, even exhaustive supervision over the entire 3 or 5-digit input domains cannot distinguish the expert algorithms from these degenerate alternatives. Furthermore, the spurious controllers need only 14 and 38 states (Sections I.1 and I.2), compared to the 11 and 32 states for the binary arithmetic experts.

The larger Softmax DFSTs in Figure 3 have enough capacity to express these spurious solutions: addition uses dimensions 22 and 33, and multiplication uses 64 and 96 (Section I.3). Nevertheless, five of six configurations in each larger sweep reach the robust testing cap. These successful configurations therefore avoid the constructed spurious rules despite sufficient capacity to represent those rules. Gradient-based optimization with the chosen parametrization and initialization repeatedly favors behavior that survives long execution horizons despite the training set failing to distinguish other spurious targets. **These results support the existence of a useful implicit bias towards generalizing rules.** Characterizing the source and reliability of this bias remains an open direction.

3.2 Length Generalization on the Universal Task

A 15-dimensional DFST with 4,335 parameters, trained on 75 PTOs from 5 programs on inputs of at most 5 symbols, reaches the 500-symbol cap on all 10 universal-simulation tasks (Figure 4). A 30-dimensional Softmax DFST with 12,720 parameters, trained on 150 PTOs, also reaches every task cap. The longest sampled trajectory requires 15,316,822 consecutive expert-action matches (Table 8).

These evaluations test 3 distinct forms of generalization: longer inputs, transfer to 5 unseen programs, and larger encoded Turing machines. Each controller keeps the same learned weights across all evaluations. Thus, a fixed model can execute novel computations without retraining, simply by changing the input Turing machine.

3.3 Controller Families and Training Dynamics

All six DFST, RNN, and GRU runs reach both binary-task caps (Figure 5). Each controller learns from 25 addition PTOs or 100 multiplication PTOs. These results support the same learning recipe across three distinct controller families: state tracking, unbounded external memory, and PTO supervision.

In Figure 5, the loss curves track fitting of the short training PTOs, while the RLG curves measure generalization to long structured inputs. All later recorded checkpoints retain the caps. Intermediate generalization need not improve monotonically, and low training loss does not certify algorithmic correctness. Section E.3 and Table 5 provide the full comparison and complete training settings.

4 Verification of Exact Learning

Beyond the finite length-generalization tests in Section 3, we extract and verify discrete controllers, then identify conditions for transferring correctness to neural execution (Section 1). Even with discrete controllers, the unbounded grid prevents a general sufficient testing horizon:

Lemma 4.1 (Undecidability of Finite-Horizon Verification)**.** *No algorithm can take descriptions of two grid agents T, M and return a finite input-length bound L such that agreement of the PTOs $T[z] = M[z]$ for every $z \in \Sigma^{\leq L}$ implies agreement on all inputs.*

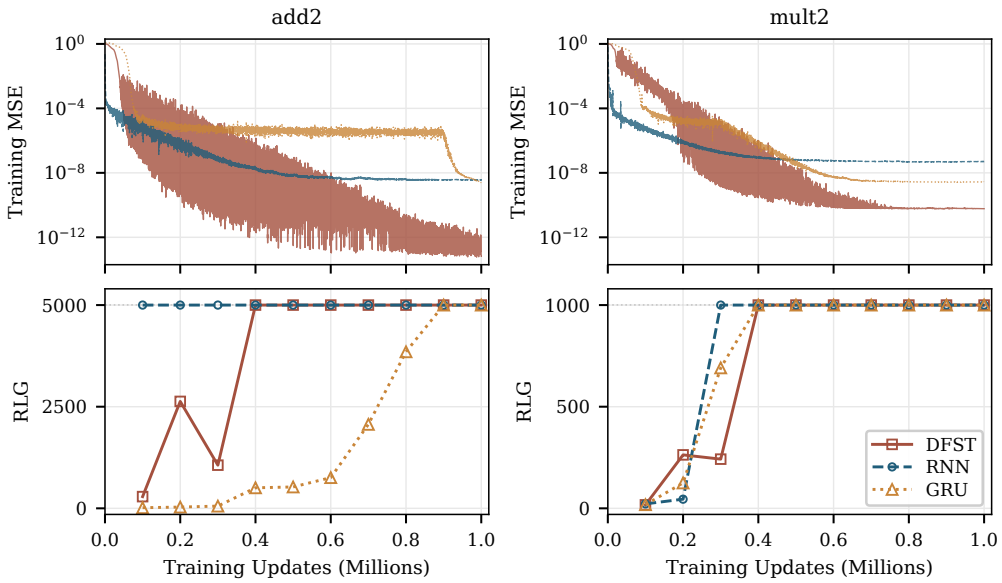


Figure 5: **Learning Dynamics across Controller Families.** All six runs reach the 5000/1000-digit caps within one million updates. Runs use different dimensions, seeds, and schedules.

Section H.1 proves the claim by reduction from the halting problem. Thus, in general, no algorithm can guarantee that length-generalizing up to a certain cap proves exact learning. To certify exact learning, we will have to consider the specific controller structure. And since finite-precision neural controllers represent prohibitively large FSTs, we first need to extract something manageable.

4.1 Recovering Explicit Controllers

Metric Extraction. We feed random inputs to a trained DFST controller and unroll the hidden-state trajectories from the executions. We connect hidden states within ε Euclidean distance and cluster the hidden states into connected components which become the initial discrete states. Each discrete state points to other discrete states through symbol transitions inherited from the hidden states inside the connected component. Thus, we have obtained a non-deterministic FST. We begin to merge the discrete states until either we get a deterministic FST and return, or until we get some inconsistency. In case of inconsistency, we halve ε and start anew. For some small enough ε, we will obtain a consistent deterministic transition graph. The resulting graph reproduces the collected trajectories, leaving unobserved transitions undefined. Interestingly, we empirically verify that after enough sampled hidden-state trajectories, the extracted graph stabilizes. Note that this type of extraction uses no expert hidden-state labels. See Section G.1 for more details.

Softmax Annealing and Direct Readout. A trained Softmax DFST assigns probabilities to successor states and actions. We continue training with a $G(p) = 1 - \|p\|_2^2$ penalty which sharpens the distributions. Progressively increasing (annealing) the penalty weight concentrates the probabilities into approximately one-hot vectors. Column-wise argmax then selects deterministic discrete transitions, thereby extracting an FST. Section G.2 gives the full objective and schedule.

Both extraction routes produce candidate discrete controllers. Figure 6 shows empirical graph stability and approximate one-hot probability concentration.

4.2 Proving Target Equivalence for the Recovered Algorithms

We relate extracted states and grid configurations to phases of the expert algorithm. The relation holds initially, and we prove that each permitted transition produces matching actions and preserves the relation. Algorithm-specific invariants show that the phases cover every valid execution. Induction then establishes agreement for arbitrary execution lengths, including simultaneous stopping. Different internal representations and state counts can therefore implement the same algorithm (Section H.3).

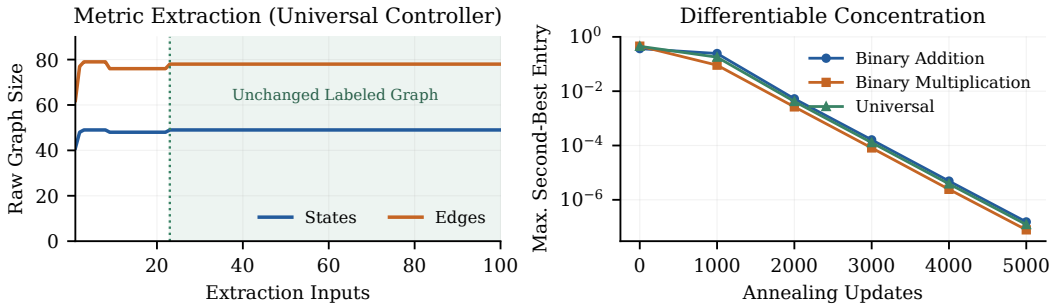


Figure 6: **From Trained Controllers to Interpretable Graphs. Left:** the extracted graph stabilizes after 23 sampled trajectories, providing finite evidence for graph stability. **Right:** runner-up probabilities approach 10^{-7} during annealing; longer annealing with larger regularization coefficients reaches the smallest positive subnormal probabilities, which remain distinct from exact zeros.

Eight Certificates. We provide five metric-extraction and three annealing certificates. Theorems H.3, H.7, H.14 and H.18 cover binary and decimal arithmetic on arbitrary nonempty operands, including unequal lengths and leading zeros. The universal certificates extend far beyond the five training programs. Theorem H.24 proves that the recovered discrete controllers implement the interpreter's rules for arbitrary correctly encoded machines and input lengths, yielding the following consequence.

Corollary 4.2 (Exact Learning of a Universal Controller)**.** *The universal controllers obtained by metric extraction and softmax annealing from differentiably trained DFSTs and Softmax DFSTs correctly simulate every binary Turing machine on every nonempty binary input.*

GPT-6 Astra generated the proofs. We verified the arguments. The repository checkers mechanically verify finite transition obligations, while the induction arguments establish unbounded correctness.

4.3 Transferring Guarantees to Neural Controllers

Both routes transfer the recovered FST certificates to neural execution under explicit assumptions.

Graph Stability for Metric Extraction. Graph stability requires every further neural hidden-state trajectory to preserve the graph extracted from trajectories S at threshold ε, up to state renaming.

Uniformly Bounded Leakage for Softmax DFSTs. We assume $\|h_t - e_{q_t}\|_1 \leq \varepsilon$ uniformly over valid inputs and execution lengths, where q_t tracks the unminimized argmax controller driven by the same observations. This bounds off-target leakage and drift from the intended one-hot state. At a one-hot hidden state, let μ denote the smallest score advantage of the prescribed write or motion over any competing choice, across all valid executions. Hidden-state leakage can reduce this advantage by at most ε. Rounding can reduce the advantage by another 2ρ, since each of the two compared scores can change by up to ρ. Thus, $\mu > \varepsilon + 2\rho$ ensures that both readouts still select the prescribed actions.

Theorem 4.3 (Behavioral Faithfulness)**.** *Under Assumption H.1, a deterministic trace-consistent metric extraction reproduces every PTO of the neural controller on every valid input. Under Assumption G.1, the Softmax DFST and the argmax controller likewise produce identical PTOs. In either case, trace-preserving quotient reduction retains this equivalence, so an expert-equivalence certificate for the extracted controller transfers to the neural controller on the same domain.*

Section H.2 proves both cases. For Softmax DFSTs, the readout margin exceeds the combined leakage and rounding perturbations, so induction preserves every action and environment update.

5 Conclusion

Our framework places differentiable parameter search inside a discrete algorithmic shell: train a neural controller, extract a discrete model through a fixed rule, and verify exact agreement with the target algorithm. Since extraction follows a fixed simple procedure (especially with argmax readout for Softmax DFSTs), we view this combination of differentiable search and discrete readout as *differentiable exact learning*. The certificates concern the recovered discrete models, while the graph-stability or bounded-leakage assumptions additionally transfer correctness to neural execution.

Acknowledgments and Disclosure of Funding

This work was partially funded by an unrestricted gift from Coefficient Giving, and the grant number 212111 from the Swiss National Science Foundation. Francesco D'Angelo was supported by the Google PhD Fellowship.

References

N. Alon, A. K. Dewdney, and T. J. Ott. Efficient simulation of finite automata by neural nets. *Journal of the ACM (JACM)*, 38(2):495–514, 1991.

M. Andrychowicz and K. Kurach. Learning efficient algorithms with hierarchical attentive memory. *arXiv preprint arXiv:1602.03218*, 2016.

D. Angluin. Learning regular sets from queries and counterexamples. *Information and computation*, 75(2):87–106, 1987.

C. Anil, Y. Wu, A. Andreassen, A. Lewkowycz, V. Misra, V. Ramasesh, A. Slone, G. Gur-Ari, E. Dyer, and B. Neyshabur. Exploring length generalization in large language models. *Advances in Neural Information Processing Systems*, 35:38546–38556, 2022.

B. D. Argall, S. Chernova, M. Veloso, and B. Browning. A survey of robot learning from demonstration. *Robotics and autonomous systems*, 57(5):469–483, 2009.

A. Back de Luca, G. Giapitzakis, and K. Fountoulakis. Learning to add, multiply, and execute algorithmic instructions exactly with neural networks. In *Advances in Neural Information Processing Systems*, volume 38, 2025. URL `https://doi.org/10.52202/085713-2636`.

S. Bhattamishra, A. Patel, and N. Goyal. On the computational power of transformers and its implications in sequence modeling. *arXiv preprint arXiv:2006.09286*, 2020.

G. E. Blelloch. Prefix sums and their applications. 1990.

M. Bošnjak, T. Rocktäschel, J. Naradowsky, and S. Riedel. Programming with a differentiable forth interpreter. In *International conference on machine learning*, pages 547–556. PMLR, 2017.

J. Cai, R. Shin, and D. Song. Making neural programming architectures generalize via recursion. *arXiv preprint arXiv:1704.06611*, 2017.

X. Chen, C. Liang, A. W. Yu, D. Song, and D. Zhou. Compositional generalization via neural-symbolic stack machines. *Advances in Neural Information Processing Systems*, 33:1690–1701, 2020.

H. Cho, J. Cha, P. Awasthi, S. Bhojanapalli, A. Gupta, and C. Yun. Position coupling: Improving length generalization of arithmetic transformers using task structure. *Advances in Neural Information Processing Systems*, 37:22233–22315, 2024a.

H. Cho, J. Cha, S. Bhojanapalli, and C. Yun. Arithmetic transformers can length-generalize in both operand length and count. *arXiv preprint arXiv:2410.15787*, 2024b.

S. Chung and H. Siegelmann. Turing completeness of bounded-precision recurrent neural networks. *Advances in neural information processing systems*, 34:28431–28441, 2021.

C. De La Higuera. Characteristic sets for polynomial grammatical inference. *Machine Learning*, 27 (2):125–138, 1997.

M. Dehghani, S. Gouws, O. Vinyals, J. Uszkoreit, and L. Kaiser. Universal transformers. *arXiv preprint arXiv:1807.03819*, 2018.

G. Delétang, A. Ruoss, J. Grau-Moya, T. Genewein, L. K. Wenliang, E. Catt, C. Cundy, M. Hutter, S. Legg, J. Veness, et al. Neural networks and the chomsky hierarchy. *arXiv preprint arXiv:2207.02098*, 2022.

A. L. Gaunt, M. Brockschmidt, R. Singh, N. Kushman, P. Kohli, J. Taylor, and D. Tarlow. Terpret: A probabilistic programming language for program induction. *arXiv preprint arXiv:1608.04428*, 2016.

C. Giles, G.-Z. Sun, H.-H. Chen, Y.-C. Lee, and D. Chen. Higher order recurrent networks and grammatical inference. *Advances in neural information processing systems*, 2, 1989.

E. M. Gold. Language identification in the limit. *Information and control*, 10(5):447–474, 1967.

E. M. Gold. Complexity of automaton identification from given data. *Information and control*, 37(3): 302–320, 1978.

A. Graves, G. Wayne, and I. Danihelka. Neural turing machines. *arXiv preprint arXiv:1410.5401*, 2014.

A. Graves, G. Wayne, M. Reynolds, T. Harley, I. Danihelka, A. Grabska-Barwińska, S. G. Colmenarejo, E. Grefenstette, T. Ramalho, J. Agapiou, et al. Hybrid computing using a neural network with dynamic external memory. *Nature*, 538(7626):471–476, 2016.

E. Grefenstette, K. M. Hermann, M. Suleyman, and P. Blunsom. Learning to transduce with unbounded memory. *Advances in neural information processing systems*, 28, 2015.

A. György, T. Lattimore, N. Lazić, and C. Szepesvári. Beyond statistical learning: Exact learning is essential for general intelligence. *arXiv preprint arXiv:2506.23908*, 2025.

H. Zhou, A. Bradley, E. Littwin, N. Razin, O. Saremi, J. Susskind, S. Bengio, and P. Nakkiran. What algorithms can transformers learn? a study in length generalization. *arXiv preprint arXiv:2310.16028*, 2023.

K. Hou, D. Brandfonbrener, S. M. Kakade, S. Jelassi, and E. Malach. Universal length generalization with Turing programs. In *Proceedings of the 42nd International Conference on Machine Learning*, volume 267 of *Proceedings of Machine Learning Research*, pages 23873–23893. PMLR, 2025. URL `https://proceedings.mlr.press/v267/hou25a.html`.

P. Indyk. Optimal simulation of automata by neural nets. In *Annual Symposium on Theoretical Aspects of Computer Science*, pages 337–348. Springer, 1995.

S. Jelassi, S. d'Ascoli, C. Domingo-Enrich, Y. Wu, Y. Li, and F. Charton. Length generalization in arithmetic transformers. *arXiv preprint arXiv:2306.15400*, 2023.

A. Joulin and T. Mikolov. Inferring algorithmic patterns with stack-augmented recurrent nets. *Advances in neural information processing systems*, 28, 2015.

L. Kaiser and I. Sutskever. Neural gpus learn algorithms. *arXiv preprint arXiv:1511.08228*, 2015.

A. Kazemnejad, I. Padhi, K. Natesan Ramamurthy, P. Das, and S. Reddy. The impact of positional encoding on length generalization in transformers. *Advances in Neural Information Processing Systems*, 36:24892–24928, 2023.

D. P. Kingma. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

K. Kurach, M. Andrychowicz, and I. Sutskever. Neural random-access machines. *arXiv preprint arXiv:1511.06392*, 2015.

N. Lee, K. Sreenivasan, J. D. Lee, K. Lee, and D. Papailiopoulos. Teaching arithmetic to small transformers. *arXiv preprint arXiv:2307.03381*, 2023.

B. Liu, J. T. Ash, S. Goel, A. Krishnamurthy, and C. Zhang. Transformers learn shortcuts to automata. *arXiv preprint arXiv:2210.10749*, 2022.

A. N. Meltzoff and M. K. Moore. Infants' understanding of people and things: From body imitation to folk psychology. *The body and the self*, pages 43–69, 1995.

W. Merrill. Sequential neural networks as automata. *arXiv preprint arXiv:1906.01615*, 2019.

W. Merrill and A. Sabharwal. The expresssive power of transformers with chain of thought. *CoRR*, 2023a.

W. Merrill and A. Sabharwal. The parallelism tradeoff: Limitations of log-precision transformers. *Transactions of the Association for Computational Linguistics*, 11:531–545, 2023b.

W. Merrill, J. Petty, and A. Sabharwal. The illusion of state in state-space models. In *Proceedings of the 41st International Conference on Machine Learning*, ICML'24. JMLR.org, 2024.

A. Neelakantan, Q. V. Le, and I. Sutskever. Neural programmer: Inducing latent programs with gradient descent. *arXiv preprint arXiv:1511.04834*, 2015.

J. Oncina and P. Garcia. Identifying regular languages in polynomial time. In *Advances in structural and syntactic pattern recognition*, pages 99–108. World Scientific, 1992.

H. Papazov and N. Flammarion. Learning algorithms in the limit. In N. Haghtalab and A. Moitra, editors, *Proceedings of Thirty Eighth Conference on Learning Theory*, volume 291 of *Proceedings of Machine Learning Research*, pages 4486–4510. PMLR, 2025.

H. Papazov, F. D'Angelo, and N. Flammarion. Exact learning of arithmetic with differentiable agents. *arXiv preprint arXiv:2511.22751*, 2025.

R. Parekh and V. Honavar. Learning dfa from simple examples. *Machine Learning*, 44:9–35, 2001.

E. Parisotto, A.-r. Mohamed, R. Singh, L. Li, D. Zhou, and P. Kohli. Neuro-symbolic program synthesis. *arXiv preprint arXiv:1611.01855*, 2016.

A. Payani and F. Fekri. Inductive logic programming via differentiable deep neural logic networks. *arXiv preprint arXiv:1906.03523*, 2019. URL `https://arxiv.org/abs/1906.03523`.

T. Pierrot, G. Ligner, S. E. Reed, O. Sigaud, N. Perrin, A. Laterre, D. Kas, K. Beguir, and N. de Freitas. Learning compositional neural programs with recursive tree search and planning. *Advances in Neural Information Processing Systems*, 32, 2019.

E. Price, W. Zaremba, and I. Sutskever. Extensions and limitations of the neural gpu. *arXiv preprint arXiv:1611.00736*, 2016.

J. Pérez, J. Marinković, and P. Barceló. On the turing completeness of modern neural network architectures. In *International Conference on Learning Representations*, 2019.

S. Reed and N. De Freitas. Neural programmer-interpreters. *arXiv preprint arXiv:1511.06279*, 2015.

D. Schuurmans, H. Dai, and F. Zanini. Autoregressive large language models are computationally universal. *arXiv preprint arXiv:2410.03170*, 2024.

R. Shen, S. Bubeck, R. Eldan, Y. T. Lee, Y. Li, and Y. Zhang. Positional description matters for transformers arithmetic. *arXiv preprint arXiv:2311.14737*, 2023.

H. T. Siegelmann. Recurrent neural networks and finite automata. *Computational intelligence*, 12(4): 567–574, 1996.

H. T. Siegelmann and E. D. Sontag. On the computational power of neural nets. In *Proceedings of the fifth annual workshop on Computational learning theory*, pages 440–449, 1992.

R. J. Solomonoff et al. Machine learning-past and future. *Dartmouth, NH, July*, 2006.

A. Svete and R. Cotterell. Recurrent neural language models as probabilistic finite-state automata. *arXiv preprint arXiv:2310.05161*, 2023.

F. Vaandrager, B. Garhewal, J. Rot, and T. Wißmann. A new approach for active automata learning based on apartness. In *International Conference on Tools and Algorithms for the Construction and Analysis of Systems*, pages 223–243. Springer, 2022.

O. Vinyals, M. Fortunato, and N. Jaitly. Pointer networks. *Advances in neural information processing systems*, 28, 2015.

C. Wei, Y. Chen, and T. Ma. Statistically meaningful approximation: a case study on approximating turing machines with transformers. *Advances in Neural Information Processing Systems*, 35: 12071–12083, 2022.

G. Weiss, Y. Goldberg, and E. Yahav. On the practical computational power of finite precision rnns for language recognition. *arXiv preprint arXiv:1805.04908*, 2018.

J. Weston, S. Chopra, and A. Bordes. Memory networks. *arXiv preprint arXiv:1410.3916*, 2014.

Y. Zhou, U. Alon, X. Chen, X. Wang, R. Agarwal, and D. Zhou. Transformers can achieve length generalization but not robustly. *arXiv preprint arXiv:2402.09371*, 2024.

W. Zaremba and I. Sutskever. Reinforcement learning neural turing machines-revised. *arXiv preprint arXiv:1505.00521*, 2015.

W. Zaremba, T. Mikolov, A. Joulin, and R. Fergus. Learning simple algorithms from examples. In *International conference on machine learning*, pages 421–429. PMLR, 2016.

H. Zhang, L. H. Li, T. Meng, K.-W. Chang, and G. V. d. Broeck. On the paradox of learning to reason from data. *arXiv preprint arXiv:2205.11502*, 2022.

Organization of the Appendix

- **Section A** positions our work within the broader literature on neural program induction, arithmetic length generalization, and learning-theoretic supervision hierarchies.
- **Section B** proves the universality theorem for DFSTs (Theorem 2.5) and discusses non-uniqueness of DFST parametrizations.
- **Section C** collects the extended notation for grid-agent execution, the hardness results motivating our learning paradigm, the pedagogical interpretation of PTOs, and the full catalog of expert agents.
- **Section D** details optimizer settings, the learning-rate schedule, per-task DFST recipes, the recurrent-controller comparisons, and the capped evaluation protocol.
- **Section E** presents additional uniform and robust generalization results, Softmax DFST sweeps, and training dynamics for DFST, RNN, and GRU controllers on binary arithmetic.
- **Section F** describes the teleportation rune mechanics, the Turing-machine encoding, and the 10 simulated tasks used in the universal-agent experiment.
- **Section G** develops metric extraction and softmax annealing, and separates empirical evidence from the assumptions needed for neural-model equivalence.
- **Section H** proves the verification obstruction and conditional faithfulness theorem, then presents eight mathematical equivalence certificates grouped by task, including the required transition maps.
- **Section I** constructs small arithmetic controllers that match every short-input PTO yet fail beyond the training lengths, and relates these counterexamples to the Softmax DFST capacity sweeps.

A Related Work

Our work builds on a long line of research on *neural program induction*, which studies the problem of learning algorithmic procedures using neural architectures. Within this paradigm, prior work has explored a wide variety of learning signals and model classes. Our framework combines the following three desiderata:

(1) Learning from expert behavior observations;
(2) Computational universality of the learning framework;
(3) Continuous optimization in a fully differentiable setting.

This work investigates exact learning at the intersection of these three principles, with equivalence proofs for extracted controllers over all valid inputs. Specifically, we focus on learning scenarios where intelligent systems infer algorithmic processes solely from externally observable behavior, as in learning-from-demonstration settings (Argall et al., 2009). Crucially, behavior observations circumvent the general identification barrier for input–output supervision (Gold, 1967; Papazov and Flammarion, 2025), without access to the expert's internal state. Furthermore, we require computational universality and differentiability from our framework in order to study learning as a general-purpose mechanism, avoiding task-specific instruction templates and hand-crafted search procedures during training.

Prior universality results differ in the computational resources and learning guarantees provided:

- **Precision and Context Requirements:** RNNs and Neural GPUs with unbounded-precision internal arithmetic (Siegelmann and Sontag, 1992; Pérez et al., 2019); Transformers with growing context windows or unbounded-precision internal arithmetic (Pérez et al., 2019; Dehghani et al., 2018; Bhattamishra et al., 2020; Merrill and Sabharwal, 2023a).
- **External-Memory Constructions:** Bounded-precision recurrent controllers coupled to growing memory (Chung and Siegelmann, 2021); language models prompted with a complete production-rule system (Schuurmans et al., 2024).
- **Horizon-Dependent Architectures:** Statistical approximation guarantees for bounded-time computation with model sizes depending on the execution horizon (Wei et al., 2022).

These results establish computational expressivity or statistical approximation rather than recovery of a persistent controller from behavior observations. Finite-precision universality alone therefore does not distinguish our framework. Our universal controller instead learns from 75 PTOs without

supplied control states or a transition table, transfers to five programs absent from training, and yields extracted-controller certificates covering arbitrary valid machine encodings (Sections F and H). The certified controllers retain the same parameters across unbounded input lengths and execution horizons.

We now compare the supervision and verification guarantees of related learning frameworks.

External-Memory Frameworks. The work of Zaremba et al. (2016) comes, perhaps, closest to our setting. The authors consider an RNN-based controller which operates over a read-only input grid and a write-only unidirectional output tape by issuing action tokens containing move and write commands. Zaremba et al. (2016) investigate whether the neural controller can learn to execute simple algorithms (e.g., copying, reversing, addition) when trained via reinforcement learning for correct outputs or with supervision from expert policies. Unlike in our experiments, training uses nontrivial curricula, and off-the-shelf Q-learning fails on most tasks without additional modifications. The task suite includes single-digit multiplication and demonstrates substantial extrapolation on selected tasks. Our experiments additionally cover unrestricted two-operand multiplication and universal simulation using an unbounded read–write workspace.

Related neurosymbolic frameworks include Higher-Order RNNs (Giles et al., 1989), Memory Networks (Weston et al., 2014), Neural Turing Machines (Graves et al., 2014), Neural GPUs (Kaiser and Sutskever, 2015), Neural Random-Access Machines (Kurach et al., 2015), Neural Programmers (Neelakantan et al., 2015), Pointer Networks (Vinyals et al., 2015), Stack-Augmented RNNs (Joulin and Mikolov, 2015), Differentiable Neural Computers (Graves et al., 2016), Hierarchical Attentive Memories (Andrychowicz and Kurach, 2016), TerpreT (Gaunt et al., 2016), R3NNs (Parisotto et al., 2016), Differentiable Forth Interpreters (Bošnjak et al., 2017), and Neural-Symbolic Stack Machines (Chen et al., 2020). External storage and even unbounded differentiable memory already appear in this literature (Grefenstette et al., 2015). Our distinction concerns learning latent controller state from PTOs and certifying the extracted computation, rather than external memory alone.

Input-Output Observations. Another closely related approach appears in Zaremba and Sutskever (2015), which considers an LSTM controller operating over a read-only input tape, a read–write memory tape, and a write-only unidirectional output tape. The addition of a read–write memory renders the resulting model computationally universal. Nevertheless, learning proceeds solely via RL with rewards for correct output. Gold (1967, Theorems I.3 and I.5) rule out identification of the full class of recursive functions from input–output observations; these results do not prohibit learning a particular arithmetic algorithm. PTOs avoid this general identification barrier (Papazov and Flammarion, 2025), while the present experiments investigate finite-data optimization and verification.

Program-Structure Observations. A particularly relevant line of work stems from Neural Programmer-Interpreters (NPIs) (Reed and De Freitas, 2015). NPIs use an LSTM controller as a program executor: At each step, the controller consumes a local environment snapshot together with the active subprogram embedding and current arguments, then outputs (i) a distribution over the next subprogram in a fixed library, (ii) the corresponding arguments, and (iii) a halt/return decision. Training traces explicitly supervise this call–return structure. Each program invocation starts a fresh recurrent state; return resumes the caller's state. These annotations expose program structure, without supplying the numerical recurrent hidden vectors.

This additional program supervision supports data-efficient learning[5] and in special cases permits formal guarantees of exactness. Cai et al. (2017) observe that when the expert program follows a recursive template in which each subprogram admits only finitely many base configurations and reductions, one can formally certify that the trained LSTM controller recovers the ground-truth algorithm. The experiments already reach 5000-digit addition, with all-input proofs for addition and several sorting tasks. Our distinction concerns recovering persistent controller state without prescribed subprogram identities or recursive boundaries, and extending the certificates to full binary and decimal multiplication and universal simulation. Pierrot et al. (2019) further demonstrate reinforcement learning through hierarchical program specifications, pre/postcondition checks, and recursive tree search; an induction argument certifies unbounded Tower of Hanoi execution for agents satisfying specified base-case and recursive-trace conditions. Our training instead uses PTO next-action losses without a supplied recursive decomposition or program library.

[5]The addition experiments train on 200 execution traces with operands of at most three digits.

Exact Instruction Learning and Logic Induction. Back de Luca et al. (2025) prove high-probability exact instruction execution in the infinite-width neural tangent kernel regime using explicit instruction templates, but *require retraining whenever the memory budget increases*; neural training experiments cover permutations, while arithmetic training remains untested because of ensemble cost. Our controllers retain fixed parameters across input lengths; the extracted-controller proofs impose no memory or execution-horizon bound.

Payani and Fekri (2019) recover explicit recursive programs from positive/negative facts and supplied background predicates, including addition for multiplication, without NPI-style program-structure supervision; the empirical studies use finite domains and provide neither length-generalization measurements nor equivalence proofs over all valid inputs. Our multiplication controllers learn digit-level actions and carry propagation without a supplied addition subroutine, and admit all-input proofs after extraction.

Arithmetic Length Generalization. Finally, we situate our contribution in the arithmetic generalization literature.

Recent empirical studies (Joulin and Mikolov, 2015; Delétang et al., 2022; Liu et al., 2022; Zhang et al., 2022; Shen et al., 2023; Kazemnejad et al., 2023; Jelassi et al., 2023) demonstrate that Transformers, modified RNNs, and LSTMs length-generalize poorly on a variety of simple algorithmic tasks when trained on input-output data. With additional data-formatting strategies (such as task-specific index hints, optimized positional encodings, input reversing, position coupling, and scratchpad guidance) multiple works on arithmetic generalization in Transformers (Anil et al., 2022; Jelassi et al., 2023; H. Zhou et al., 2023; Lee et al., 2023; Y. Zhou et al., 2024; Hou et al., 2025; Cho et al., 2024a,b) achieve substantial finite-length extrapolation. Most notably, for decimal addition, Y. Zhou et al. (2024) report above 98% exact-match accuracy (EMA) on 100-digit summands after training on decimal addition with up to 40-digit numbers (2.5× LG); and Cho et al. (2024a) report above 95% EMA on roughly 6–7× longer operands, with a separate scale-up experiment reaching 500 digits after training on operands of up to 160 digits. The clearest quantitative distinction concerns *both operands* in decimal multiplication: Cho et al. (2024b) train on operands of at most 10×10 digits and attain at least 78.55% median EMA through 20×15 digits, counting the entire response including the scratchpad, whereas Hou et al. (2025) extrapolate from 50×1 or 50×3 digits to 100×1 or 100×3 digits with 97% accuracy. Our selected decimal-multiplication controller trains on operands of at most 5×5 digits and passes every action-level probe at 1000×1000 digits before extraction; the extracted controller additionally admits a proof over all valid inputs.

Interestingly, similar to us, Hou et al. (2025) train on scratchpad data consisting of the computational traces of a Turing machine (TM) designed to solve the specific algorithmic task. However, whereas our agent only observes the evolution of the environment, Hou et al. (2025) supervise explicit machine states, head positions, and tape contents. The random-machine experiments evaluate one-step prediction on longer tapes, without establishing correct execution over unbounded trajectories. The accompanying RASP construction guarantees $\exp(n)$ simulated steps with a program of size $O(n)$ under distinct-n-gram and in-memory execution assumptions; this construction does not certify experimentally trained weights. Our PTOs withhold expert control states, and the extracted-controller proofs impose no input-length or execution-horizon bound.

We conclude by comparing our results in Table 1 against the Neural GPU (Kaiser and Sutskever, 2015; Price et al., 2016), which achieves strong arithmetic generalization from input-output observations. Kaiser and Sutskever (2015) introduce a Neural Cellular Automaton, called the Neural GPU, which achieves remarkable random-test generalization when trained solely on input-output examples (and an upper bound on the computation length) of binary addition and multiplication. Error-free sampled binary multiplication already reaches 2000 bits, exceeding our 1000-digit testing cap. However, Price et al. (2016) report sensitivity to the random seed and document long-carry and symmetric-input failures, despite greater than 99% whole-answer accuracy for decimal multiplication at 100 digits. Our selected models pass all repdigit cross-pairs at the testing caps in both bases, including decimal multiplication, and the extracted controllers admit proofs of exact learning. The single-seed DFST schedule sweeps do not establish superior robustness across random seeds. Furthermore, our DFST agents train on small PTO datasets using Adam, gradient clipping, and a learning-rate schedule, while Neural GPU training combines curricula, parameter-sharing relaxation, dropout, and gradient noise. As shown in Table 1, observable intermediate steps support extreme generalization with far fewer training examples, although PTOs provide richer supervision than input-output pairs. RNN and GRU

runs also reach both binary-task caps, supporting the broader state-tracking recipe beyond DFSTs (Section E.3).

Table 1: Neural GPU vs. DFST on Arithmetic Tasks. DFST entries report the selected runs before extraction; lengths count bits in binary and digits in decimal.

Framework	Metric	`add2`	`add10`	`mult2`	`mult10`
Neural GPU with Input–Output Data	# Samples	$\sim$ 200k	–	$\sim$ 200k	–
	# Parameters	10368	–	10368	–
	Training Digits	20	–	20	–
	RLG	≤ 20	–	≤ 8	–
	ULG	2000	–	2000	–
DFST (ours) with PTOs	# Samples	25	250	100	10000
	# Parameters	891	8227	5280	162108
	Training Digits	3	3	5	5
	RLG	≥ 5000	≥ 5000	≥ 1000	≥ 1000
	ULG	≥ 5000	≥ 5000	≥ 1000	≥ 1000

Summary. Our contribution combines latent-state discovery from behavior, extreme extrapolation in both arithmetic operands, and exact verification within one universal framework. Eight certificates cover extracted controllers across all five trained tasks. Graph stability or a suitable invariant of the annealed dynamics additionally transfers these guarantees to the neural controllers (Sections G and H).

B Differentiable Finite-State Transducers

This appendix collects the formal details of DFST expressivity deferred from Section 2. We prove Theorem 2.5, discuss the non-uniqueness of emulating parametrizations, and relate the required hidden-state dimension to the number of states of the target agent.

B.1 Proof of Universality

We restate the theorem for convenience.

Theorem 2.5 (DFST-Grid Universality)**.** *Any d-state grid agent admits exact emulation by a d-dimensional DFST at any finite arithmetic precision, and by a $(d+1)$-dimensional Softmax DFST with finite logits in exact real arithmetic. A d-dimensional Softmax DFST also suffices under rounded arithmetic whose softmax attains simplex vertices and whose recurrence preserves vertex operations.*

Proof. Let $T = (Q, \Sigma, \Gamma, \delta, \gamma, q_0, q_f) \in \mathcal{A}$ denote a grid agent with d states $q_0, \ldots, q_{d-1} = q_f$. Write $k = |\Sigma|$ for the alphabet size with symbols $s_0, \ldots, s_{k-1}$, and $r = |\mathcal{M}|$ for the number of motion tokens $m_0, \ldots, m_{r-1}$. We encode states, symbols, and motions as standard basis vectors: $q_i \leftrightarrow e_{i+1} \in \mathbb{R}^d$, $s_j \leftrightarrow e_{j+1} \in \mathbb{R}^k$, and $m_\ell \leftrightarrow e_{\ell+1} \in \mathbb{R}^r$.

Define the DFST weights (A, B, C, h_0) as follows. Set $h_0 = e_1 = (1, 0, \ldots, 0)^\top \in \mathbb{R}^d$. For each transition $\delta(q_i, s_j) = q_{i'}$ with $\gamma(q_i, s_j) = (s_{j'}, m_\ell)$, assign

$$A[j,\, i',\, i] = 1, \qquad B[j,\, j',\, i] = 1, \qquad C[j,\, \ell,\, i] = 1,$$

and set all remaining tensor entries to 0.

Under this assignment, if $h_{t-1} = e_{i+1}$ encodes state q_i and the input $x_t = e_{j+1}$ encodes symbol s_j, then $A[x_t]\, h_{t-1} = e_{i'+1}$, $B[x_t]\, h_{t-1} = e_{j'+1}$, and $C[x_t]\, h_{t-1} = e_{\ell+1}$. Since $h_0 = e_1$ encodes the initial state q_0, induction on t shows that the DFST(A, B, C, h_0) exactly reproduces the output sequence of T on every input. This construction uses only $\{0, 1\}$-valued weights and hence works at any finite arithmetic precision $p \geq 1$.

Softmax DFSTs in Exact Real Arithmetic. Let $D_x \in \{0, 1\}^{d \times d}$ denote the deterministic transition matrix for read symbol x, and let $O^s_x \in \{0, 1\}^{k \times d}$ and $O^m_x \in \{0, 1\}^{r \times d}$ denote the corresponding

one-hot output matrices. Set $n = d + 1$ and $u = \mathbf{1}_n/n$. Define

$$V = \begin{pmatrix} I_d \\ -\mathbf{1}_d^\top \end{pmatrix}, \qquad W = (I_d \quad 0) - \mathbf{1}_d u^\top.$$

Then $WV = I_d$, $Wu = 0$, and $\mathbf{1}_n^\top V = 0$. Choose $0 < \alpha < 1$ and $\eta > 0$ sufficiently small that every entry of

$$A[x] = u\mathbf{1}_n^\top + \alpha V D_x W, \qquad h_0 = u + \eta V e_{q_0}$$

remains strictly positive. Such choices exist because the alphabet and state set contain finitely many elements. The columns of $A[x]$ and the vector h_0 sum to one. Induction gives

$$h_t = u + \eta\alpha^t V e_{q_t}, \qquad q_t = \delta(q_{t-1}, x_t).$$

For either output alphabet of size $a \in \{k, r\}$, write $u_a = \mathbf{1}_a/a$, $V_a = I_a - u_a\mathbf{1}_a^\top$, and $O_x \in \{O_x^s, O_x^m\}$. Choose $\beta > 0$ sufficiently small that the column-stochastic matrix

$$P[x] = u_a\mathbf{1}_n^\top + \beta V_a O_x W$$

has strictly positive entries, and use $P = B$ or $P = C$ accordingly. If y_t denotes the expert output on (q_{t-1}, x_t), the readout satisfies

$$P[x_t]h_{t-1} = u_a + \beta\eta\alpha^{t-1}(e_{y_t} - u_a).$$

The coordinate y_t therefore attains the unique maximum whenever $a > 1$; a singleton output alphabet needs no comparison. Every strictly positive probability vector admits finite logits given by the coordinatewise logarithm. Thus these tensors and initialization define a Softmax DFST reproducing every expert action at every finite time. The margins shrink with t, so this exact-real construction alone supplies no finite-precision robustness guarantee.

Softmax DFSTs under Rounded Arithmetic. For the d-dimensional one-hot construction above, assign logit 0 to each selected coordinate and $-L$ to every other coordinate. A stable softmax whose exponential rounds $\exp(-L)$ to zero produces exactly the required one-hot column; apply the same construction to the initial state. For example, correctly rounded binary32 exponentiation with round-to-nearest underflows $\exp(-128)$ to zero. Once the implementation produces these vertices and preserves the corresponding matrix–vector operations, the original induction applies unchanged for every execution length. The saturation condition concerns the effective probabilities, rather than large finite logits alone.

For Turing-completeness, observe that standard Turing machines arise as grid agents restricted to a single row of the grid. Each construction emulates every such agent under the stated arithmetic conditions, so both controller families can simulate any Turing machine. □

B.2 Non-Uniqueness of Emulating Parametrizations

The one-hot construction in the proof above produces one particular DFST that emulates a given grid agent T, but many other parametrizations achieve the same behavior. In exact arithmetic, any invertible matrix $R \in \mathbb{Q}^{d\times d}$ defines an equivalent controller through

$$A'[x] = RA[x]R^{-1}, \qquad B'[x] = B[x]R^{-1},$$
$$C'[x] = C[x]R^{-1}, \qquad h_0' = Rh_0.$$

Induction gives $h_t' = Rh_t$ for every read sequence, while $B'[x_t]h_{t-1}' = B[x_t]h_{t-1}$ and $C'[x_t]h_{t-1}' = C[x_t]h_{t-1}$ preserve both readouts. Thus the transformed controller reproduces every action and PTO. Choosing $R = cI_d$ for distinct nonzero rational values c already yields infinitely many distinct exact-arithmetic parametrizations, since $h_0 \neq 0$. These transformations preserve the PTO loss and therefore generate infinitely many zero-loss realizations of the expert's one-hot targets.

A fixed finite arithmetic format admits only finitely many parameter configurations. Arbitrary rational changes of basis need not preserve representability or execution under rounding, so the infinitude claim does not extend to fixed finite precision. Permutations of the one-hot construction nevertheless preserve the binary weights and vertex operations, yielding equivalent finite-precision realizations whenever those operations remain exact. Separately, adding unreachable states yields infinitely many discrete grid agents with the same observable policy.

B.3 Hidden-State Dimension and State Complexity

Theorem 2.5 guarantees that an n-dimensional ordinary DFST suffices to emulate any n-state grid agent. Most ordinary-DFST experiments therefore match the hidden dimension to the expert state count, but successful learning does not require an exact match. For the 11-state `add2` expert, a sweep over hidden dimensions $d \in \{5, 8, 11, 14, 17\}$ reaches the 5000-digit RLG cap at every tested dimension except $d = 5$. Thus both a smaller controller and controllers with excess capacity achieve extreme tested generalization. These finite probes do not establish exact simulation on arbitrary inputs or identify the minimum sufficient dimension.

The Softmax DFST sweeps provide complementary evidence for the benefits of additional capacity (Figures 9 and 13). Binary addition reaches both caps with dimensions 22 and 33, compared with 11 expert states; binary multiplication reaches both caps with dimensions 64 and 96, compared with 32 expert states. Universal simulation reaches every task cap with dimensions 30 and 45, compared with 15 expert states. Across these tasks, successful configurations therefore accommodate two or three times the expert state count. Larger dimensions do not improve every configuration, but these results support choosing excess capacity when the expert state count remains unknown: precise knowledge of that count need not precede successful training.

Precisely characterizing the minimum hidden dimension required for a DFST to simulate a given n-state FST remains an open direction. The one-hot construction supplies an upper bound of n for ordinary DFSTs, while smaller representations may exploit the structure of the target transitions and outputs. Such a characterization should distinguish agreement on all input strings from agreement only on realizable environment traces, and account for arithmetic precision and readout margins. A separate question concerns whether gradient-based training can reliably discover representations near the minimum dimension.

C Grid Agents, PTOs, and Task Catalog

This appendix collects the pedagogical interpretation of PTOs, the formal notation for grid-agent execution, the hardness results that motivate our learning paradigm, and the full catalog of expert agents used in our experiments.

C.1 Pedagogical Interpretation of PTOs

Intelligent agents rarely acquire novel algorithmic skills from input–output pairs alone. Rather, learning typically proceeds from observing the evolution of an underlying computational process and inferring the target function from intermediate steps. The PTO formalism admits a concrete pedagogical interpretation: A grid agent models a teacher navigating a blackboard, with q_t corresponding to the teacher's mental state, p_t marking the chalk's position, and x_{t+1} denoting the symbol written at that position. Students observe the chalk's movements m_{t+1} and the evolving blackboard contents s_{t+1} but lack access to the teacher's neuronal activity $q_t \to q_{t+1}$. PTOs thus capture a rich but reasonable chain of thought – not unlike the algorithmic scratchpad data employed to boost length generalization in Transformers (Lee et al., 2023), yet grounded in the natural structure of agentic computation rather than requiring internal-state annotations (Hou et al., 2025; Cai et al., 2017).

C.2 Extended Notation for Grid-Agent Execution

We extend δ and γ recursively to $Q \times \Sigma^\star$ in the standard way and denote by $T(x) = T(x_1 \dots x_n)$ the output sequence $\gamma(q_0, x_1)$, $\gamma(\delta(q_0, x_1), x_2)$, …, $\gamma(\delta(q_0, x_1 \dots x_{n-1}), x_n)$.

Each algorithmic task $\mathcal{P}$ comes with a set of well-specified inputs $S_\mathcal{P} \subseteq \Sigma^\star$. For arithmetic tasks, each input $x \in S_\mathcal{P}$ occupies cells $(0, 0), \dots, (|x| - 1, 0)$ on an otherwise empty grid. Universal inputs use the program-and-tape encoding in Section F.2. We also denote the initial world-state by x (overloading notation). Running the expert grid agent T on x produces a PTO $T[x]$. We write $T\{x\}$ for the *input trace* – the sequence of symbols observed by T during execution on x – so that $T(T\{x\})$ gives the corresponding output trace.

For a training set $S \subseteq S_\mathcal{P}$, we write $T[S]$ for the PTOs induced by S and $T\{S\}$ for the input traces due to S. Let $I = T\{S_\mathcal{P}\}$ denote the set of all realizable input traces.

C.3 Hardness of Learning from Restricted Input Traces

Since I constitutes a restricted subset of $\Sigma^\star$, a training set S – no matter how large – may leave some transitions of T unexplored. This incompleteness makes the learning problem fundamentally hard: Many irreducible FSTs can exhibit identical behavior on I.

Learning FSTs from complete data constitutes a classical problem in grammatical inference, enjoying polynomial-size characteristic sets (Gold, 1978; Oncina and Garcia, 1992; De La Higuera, 1997; Parekh and Honavar, 2001) and efficient active-learning algorithms (Angluin, 1987; Vaandrager et al., 2022). However, restricting all observable inputs to the unknown set I dramatically increases difficulty. In particular, Papazov and Flammarion (2025, Theorem 17) proved that no polynomial-size characteristic set exists for this restricted problem, implying that no algorithm can learn the policy of every grid agent from polynomially many PTOs with lengths polynomial in the number of states of the expert agent.

C.4 Expert Agent Catalog

Table 2 summarizes the state counts, alphabets, and runtimes of all grid agents and Turing machines used in our experiments. For arithmetic tasks, m and n denote the lengths in digits of the two operands; for all other tasks, n denotes the input length.

For universal simulation, Q_M denotes the state set of the simulated Turing machine M, including the halting state, and $|M[z]|$ counts the simulated machine's actions on input z, including the stopping action. The runtime bound counts each teleportation as one action.

The public controller walkthrough provides animated expert executions and instructions for inspecting individual computation steps.

Table 2: Grid Agents and Turing Machines Used in Experiments.

Task	# States $\lvert Q\rvert$	Alphabet Σ	Runtime
Grid Agents			
`add2`	11	$\{0, 1, \lambda, +\}$	$O((m+n)^2)$
`add10`	19	$\{0, \ldots, 9, \lambda, +\}$	$O((m+n)^2)$
`mult2`	32	$\{0, 1, \lambda, \times\}$	$O((m+n)^2)$
`mult10`	108	$\{0, \ldots, 9, \lambda, \times\}$	$O((m+n)^2)$
Turing Machines (simulated by U)			
`parity`	3	$\{0, 1, \lambda\}$	$\Theta(n)$
`flip`	2	$\{0, 1, \lambda\}$	$\Theta(n)$
`cum-sum`	3	$\{0, 1, \lambda\}$	$\Theta(n)$
`shift-left`	5	$\{0, 1, \lambda\}$	$\Theta(n)$
`shift-right`	7	$\{0, 1, \lambda\}$	$\Theta(n)$
`scaling-A`(s)	$s+2$	$\{0, 1, \lambda\}$	$\Theta(s)$
`scaling-B`(s)	$4s+1$	$\{0, 1, \lambda\}$	$\Theta(s)$
`bit-delete`	5	$\{0, 1, \lambda\}$	$\Theta(n)$
`reverse`	16	$\{0, 1, \lambda\}$	$\Theta(n^2)$
`copy`	10	$\{0, 1, \lambda\}$	$\Theta(n^2)$
Universal Agent			
$U(\langle M, z\rangle)$	15	$\{0, 1, \lambda, X, Y, Z, \texttt{L}, \texttt{R}, \texttt{S}\}$	$O(\lvert Q_M\rvert \, \lvert M[z]\rvert)$

D Implementation and Experimental Details

We specify the data, optimization, execution semantics, and evaluation protocol for the experiments. Section A compares these results with prior work.

Table 3: **Generalization before Extraction.** Selected ordinary DFST runs; N counts training PTOs and q bounds training operand length. Caps indicate successful finite probes, without asserting a failure beyond the cap.

Task	d	N	q	Updates	Warm-up	ULG / RLG
`add2`	11	25	3	500k	50k	$\geq 5000/\geq 5000$
`add10`	19	250	3	4M	50k	$\geq 5000/\geq 5000$
`mult2`	32	100	5	500k	25k	$\geq 1000/\geq 1000$
`mult10`	108	10 000	5	4M	0	$\geq 1000/\geq 1000$

The public repository provides training recipes, saved model weights and run configurations, and commands for reproducing the figures and analyses.

D.1 Data and Controller Initialization

Arithmetic training uses small PTO datasets from distinct operand-string pairs, retaining leading zeros. Each set includes all single-digit pairs. Remaining pairs follow independent uniform sampling of each operand length from $1, \ldots, q$ and of each digit from $0, \ldots, R-1$, with duplicate pairs rejected. The ordinary-DFST datasets contain 25, 250, 100, and 10 000 PTOs for `add2`, `add10`, `mult2`, and `mult10`, respectively. Addition uses $q = 3$ and multiplication uses $q = 5$. These experiments use the ordinary sampler without an additional teaching-set coverage core. The plotted DFST and Softmax DFST sweeps use data seed 42. Separately, we replicated strong length generalization across three data-generation seeds using the same deterministic initialization recipe.

Minibatch sampling chooses uniformly among occupied operand-length buckets, then uniformly among PTOs within the chosen bucket. Universal training first chooses a represented training task uniformly once per minibatch, then samples length buckets within that task. Consequently, training does not sample uniformly over all stored PTOs.

Ordinary DFSTs start from $A[x] = I_d$, $B[x] = 0$, $C[x] = 0$, and a trainable $h_0 = \mathbf{1}_d$. In the ordinary-DFST schedule sweeps, hidden dimensions match the experts: 11, 19, 32, and 108 for the four arithmetic tasks, and 15 for universal simulation. Inputs use frozen one-hot embeddings. The resulting arithmetic controllers contain 891, 8227, 5280, and 162 108 trainable parameters, respectively. Training uses `float32` arithmetic and logarithmic-depth prefix scans.

All DFST and Softmax DFST models begin PTO training from scratch without a pretrained checkpoint. Softmax DFSTs initialize the transition logits to I_d, output logits to zero, and initial-state logits to $\mathbf{1}_d$, then apply softmax normalization. Consequently, the effective initial state and output columns start uniformly, while transition columns favor the identity successor. Ordinary and softmax universal controllers contain 4335 and 12 720 trainable parameters at dimensions 15 and 30, respectively.

D.2 Optimizer and Learning-Rate Schedule

Ordinary DFST training uses Adam (Kingma, 2014) with $(\beta_1, \beta_2) = (0.9, 0.999)$, $\epsilon = 10^{-8}$, zero weight decay, batch size 32, and global gradient-norm clipping at 1. Let T denote the update budget, ω the warm-up duration, c the constant-rate duration, γ the peak rate, and γ_f the floor, with $\omega + c < T$. The schedule follows

$$\gamma(t) = \begin{cases} \gamma_f^{1-t/\omega}\gamma^{t/\omega}, & 0 \leq t < \omega, \\ \gamma, & \omega \leq t < \omega + c, \\ \gamma^{1-(t-\omega-c)/(T-\omega-c)}\gamma_f^{(t-\omega-c)/(T-\omega-c)}, & \omega + c \leq t \leq T. \end{cases} \tag{1}$$

A zero-duration phase contributes no updates. The DFST sweeps use $c = 0$; GRU dynamics include the plateaus in Table 5. Updates use $t = 0, \ldots, T-1$; the endpoint at T equals the floor. Within the DFST sweeps, binary arithmetic uses $\gamma = 10^{-2}$; decimal arithmetic and universal simulation use $\gamma = 10^{-3}$. The DFST sweeps use $\gamma_f = 10^{-7}$; the separate multiplication dynamics run uses 10^{-9}.

Table 4: Ordinary-DFST Schedule Sweeps. Every Cartesian-product combination contributes one cell in the arithmetic plots. Universal simulation varies the training-set size and update budget.

Task	Hidden dimension	Total updates T	Warm-up ω
`add2`	11	$\{250\text{k}, 500\text{k}, 1\text{M}\}$	$\{25\text{k}, 50\text{k}, 100\text{k}\}$
`mult2`	32	$\{250\text{k}, 500\text{k}, 1\text{M}\}$	$\{25\text{k}, 50\text{k}, 100\text{k}\}$
`add10`	19	$\{1\text{M}, 2\text{M}, 4\text{M}\}$	$\{0, 50\text{k}, 100\text{k}\}$
`mult10`	108	$\{1\text{M}, 2\text{M}, 4\text{M}\}$	$\{0, 50\text{k}, 100\text{k}\}$
Universal	15	$\{250\text{k}, 500\text{k}\}$	0

D.3 Loss and Training Loop

For a minibatch, let M_t mask valid timesteps and let $\mathbf{1}_{s_t}$ and $\mathbf{1}_{m_t}$ encode the expert write and motion actions. The masked two-head MSE reads

$$\mathcal{L} = \frac{\sum_t M_t \big(\|\hat{s}_t - \mathbf{1}_{s_t}\|_2^2 + \|\hat{m}_t - \mathbf{1}_{m_t}\|_2^2 \big)}{2 \sum_t M_t}. \tag{2}$$

The sum extends over the batch and time axes. Padding contributes neither recurrent updates nor loss. Training feeds expert observations into the controller and supervises the next write and motion. The ordinary-DFST runs use no curriculum, inverse-frequency weighting, or extraction during optimization. Logged losses average the action objective over updates in each logging window. The training-dynamics figure averages loss over 1000 updates and evaluates saved checkpoints every 100 000 updates.

D.4 Softmax DFST Training

The probability parametrization in Section G replaces each transition and output column by a softmax distribution and normalizes the initial state. The initial training sweep uses 100 000 Adam updates at constant learning rate 0.1, MSE, batch size 32, gradient clipping at 1, and zero regularization weight. For `add2`, dimensions $\{11, 22, 33\}$ combine with dataset sizes $\{25, 50, 75\}$; for `mult2`, dimensions $\{32, 64, 96\}$ combine with sizes $\{100, 200, 300\}$. Universal simulation uses dimensions $\{15, 30, 45\}$ and sizes $\{75, 150, 225\}$. The later annealing stage follows the separate procedure in Section G. Neither the softmax sweep nor annealing shares the ordinary-DFST learning-rate schedule by default.

D.5 Additional State-Tracking Controllers

All controller families receive the currently observed symbol and emit separate write and motion scores. RNN and GRU heads act on the post-update hidden vector; DFST heads act on the previous vector together with the current observation. An Elman RNN uses

$$h_t = \tanh(W_x x_t + W_h h_{t-1} + b), \qquad \hat{s}_t = W_s h_t + b_s, \quad \hat{m}_t = W_m h_t + b_m.$$

GRU controllers replace the recurrence by gated reset, update, and candidate-state operations while retaining the same action heads. All recurrent baselines use one layer and frozen one-hot input embeddings. The RNN addition run initializes recurrent weights orthogonally, input weights with Xavier initialization, and recurrent biases to zero, with a fixed zero initial state. The RNN multiplication run and both GRU runs use zero input weights, zero recurrent biases, and a trainable all-ones initial state. For RNN multiplication, the recurrent matrix starts at the identity. For GRUs, only the candidate-state recurrent block starts at the identity; reset- and update-gate recurrent blocks start at zero. All RNN and GRU action heads use the linear layers' default random initialization.

Addition controllers contain 891, 1513, and 3977 trainable parameters for DFST, RNN, and GRU, respectively; multiplication controllers contain 5280, 5129, and 14089. Each addition dataset contains 25 PTOs with operands of at most three digits; each multiplication dataset contains 100 PTOs with operands of at most five digits. The different seeds produce different datasets within the same sampling recipe. Consequently, the training histories demonstrate successful learning across controller families without isolating architecture from initialization, data, and optimization choices.

Task / controller	d	γ	γ_f	ω	c	Clip	Seed
`add2` / DFST	11	10^{-2}	10^{-7}	50k	0	1	42
`add2` / RNN	32	10^{-2}	10^{-7}	0	0	0.1	123
`add2` / GRU	32	$3 \cdot 10^{-3}$	10^{-7}	100k	800k	0.1	17
`mult2` / DFST	32	10^{-2}	10^{-9}	25k	0	1	42
`mult2` / RNN	64	$3 \cdot 10^{-3}$	10^{-7}	0	0	0.1	1
`mult2` / GRU	64	$3 \cdot 10^{-3}$	10^{-7}	100k	200k	0.1	11

Table 5: **Controller-Comparison Training Configurations.** Every run uses $T = 10^6$ Adam updates, MSE, batch size 32, zero weight decay, and no curriculum or inverse-frequency weighting. Warm-up and plateau durations count updates; cooldown lasts $T - \omega - c$. Clip denotes the global gradient-norm threshold. The seed controls dataset generation, model initialization, and training sampling; evaluation uses seed 42 for every run.

D.6 Length-Generalization Evaluation

Every probe streams an expert PTO and requires the predicted write *and* motion to match at every real timestep, including the single final stopping action. ULG tests ten uniformly sampled pairs with both operands of exactly L digits. RLG additionally tests all R^2 repdigit pairs (a^L, b^L) with $a, b \in \{0, \ldots, R-1\}$: four pairs for binary arithmetic and 100 for decimal arithmetic. A deterministic generator indexed by the evaluation seed, task, and candidate length fixes random probes independently of the search history.

The search first tests the cap: 5000 digits for addition, 1000 for multiplication, and 500 input symbols for universal simulation. A successful cap probe terminates the search. Otherwise, binary search probes smaller lengths; RLG cannot exceed the measured ULG. Binary search assumes monotone pass/fail behavior with length and does not exhaustively test every shorter length. Consequently, reported values describe the implemented search, and the notation $\geq L_{\max}$ records a successful capped test, not an exact maximal horizon or a mathematical guarantee for all shorter inputs.

Evaluation uses batches of at most ten expert streams, chunks of 8192 transitions, and continuation state across chunks. Ordinary DFST evaluations, including the training-dynamics runs, use parallel matrix scans; the initial softmax evaluations use sequential recurrence. RNN and GRU evaluations retain the native recurrent implementation with continuation states across chunks, even when the shared evaluation configuration records the parallel scan option. These choices matter under finite precision: parallel and sequential products can differ through rounding, and ordinary-DFST sequential stabilization rescales hidden states while the parallel path uses unscaled products. All statements about agreement concern the recorded execution mode. Extraction and the conditional transfer arguments require the corresponding execution semantics rather than an unqualified interchange of numerical implementations.

Teacher Forcing and Closed-Loop Agreement. For a deterministic recurrent execution rule and environment, exact next-action agreement along the complete expert PTO implies identical closed-loop execution on that input: induction gives the same environment, next observation, and controller history after every matching action. Floating-point parallel scans require additional care because the arithmetic association can differ from a stepwise rollout. We report the actual teacher-forced exact-match protocol and treat the autonomous extracted-FST equivalence proofs separately.

Compute Reporting. The released configurations specify CUDA execution on single NVIDIA A100-SXM4- 40GB and Tesla V100-SXM2-32GB GPUs.

D.7 Optimization Limitations and Dataset Coverage

The ordinary-DFST sweep reaches the RLG cap in five of nine `add2`, three of nine `mult2`, five of nine `add10`, and two of nine `mult10` configurations. Additional updates need not improve RLG, and low training loss alone does not certify algorithmic correctness. The results demonstrate successful configurations across tasks and parametrizations, without asserting success for every schedule or seed.

Task	Controller	First ULG cap	First RLG cap	Final ULG / RLG
`add2`	DFST	200k	400k	$\geq 5000/\geq 5000$
	RNN	100k	100k	$\geq 5000/\geq 5000$
	GRU	500k	900k	$\geq 5000/\geq 5000$
`mult2`	DFST	400k	400k	$\geq 1000/\geq 1000$
	RNN	300k	300k	$\geq 1000/\geq 1000$
	GRU	400k	400k	$\geq 1000/\geq 1000$

Table 6: **First Recorded Cap-Reaching Checkpoints and Final Results.** Checkpoints occur every 100k updates; first recorded success does not identify the exact crossing time. Final results refer to one million updates. Caps bound the tests rather than the algorithms.

Our Softmax DFST training attempts on decimal addition and multiplication failed to fit even very small training datasets. The softmax distributions quickly saturate, gradients vanish, and learning stalls, consistent with convergence to a poor local minimum. These failures concern fitting the training examples, before length generalization or extraction. Developing a robust training procedure for larger Softmax DFST controllers remains an open direction for future experiments.

Dataset coverage also affects generalization. Without guaranteed transition coverage, finite random sampling can omit a reachable expert transition with nonzero probability, leaving the corresponding behavior unrepresented in the training PTOs. An important research direction concerns identifying minimal teaching sets that cover all reachable expert transitions and provide sufficient information for exact learning and extreme length generalization. Transition coverage alone need not distinguish the expert from every alternative controller, so this objective also requires understanding which execution contexts resolve behavioral ambiguities. Our repository implements preliminary teaching sets that exercise all reachable transitions of the task experts, but we have not yet evaluated these sets for neural-controller training.

E Additional Generalization Results

E.1 Uniform and Robust Generalization

Figures 7 and 8 present the complete ULG and RLG sweeps for ordinary DFSTs. Uniform success can coexist with failures on structured repdigit inputs: for example, every `add2` run reaches ULG 5000, while only five of nine runs reach RLG 5000. The two measurements therefore reveal distinct failure modes.

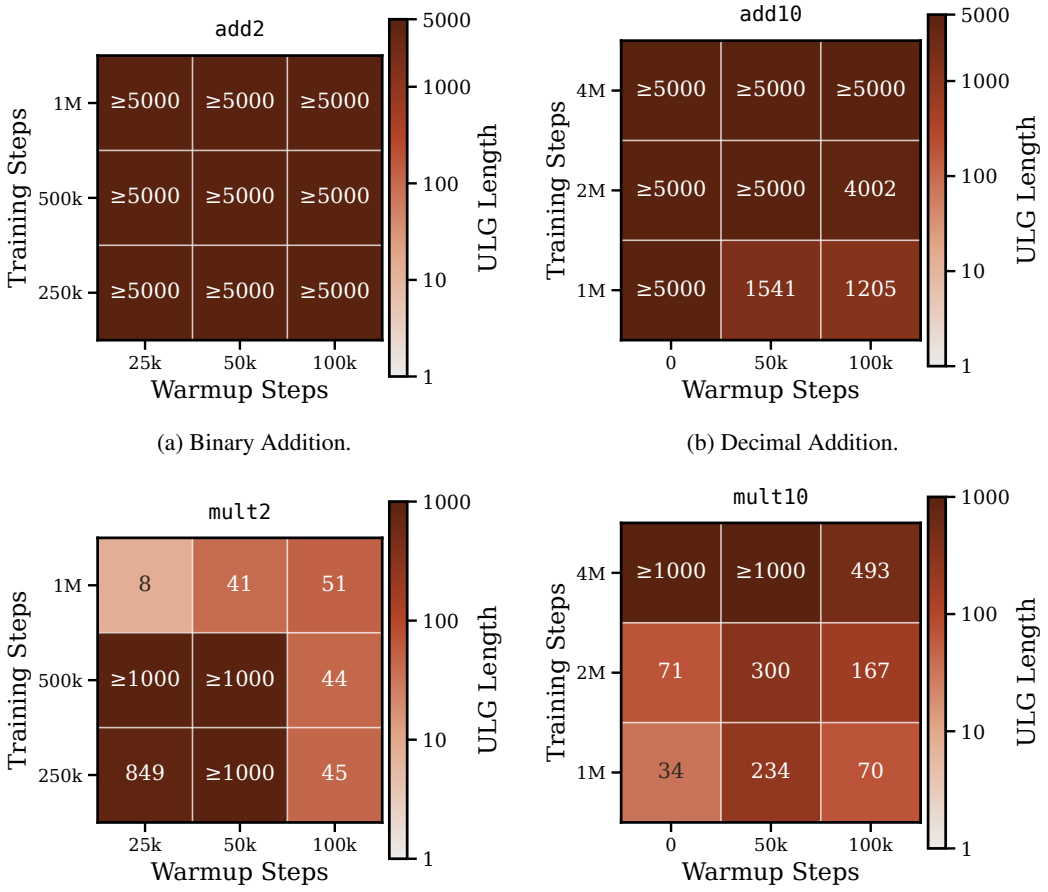
E.2 Softmax Parametrization

Figure 9 reports the initial differentiable-training sweep, before annealing or argmax extraction. Five of nine binary-addition configurations reach both caps. Five of nine binary-multiplication configurations also reach both caps, providing starting points for the interpretable extraction procedure in Section G. Every recorded cell gives identical ULG and RLG, although ordinary DFST sweeps exhibit gaps between the two metrics. Increased state dimension and additional data can help optimization, but the sweep does not support monotone improvement with either quantity. The cap-reaching checkpoints use unregularized training; sharpening follows as a separate continuation stage.

E.3 Training Dynamics across Controller Families

The released training histories include loss logs, run configurations, and checkpoint evaluations for all six runs below.

Figure 10 compares million-update DFST, RNN, and GRU runs on both binary tasks. Each addition run uses 25 training PTOs with operands of at most three digits; each multiplication run uses 100 PTOs with operands of at most five digits. All six final checkpoints reach ULG and RLG caps of 5000 for addition and 1000 for multiplication. RLG tests all four binary repdigit cross-pairs.



(a) Binary Addition.

(b) Decimal Addition.

(c) Binary Multiplication.

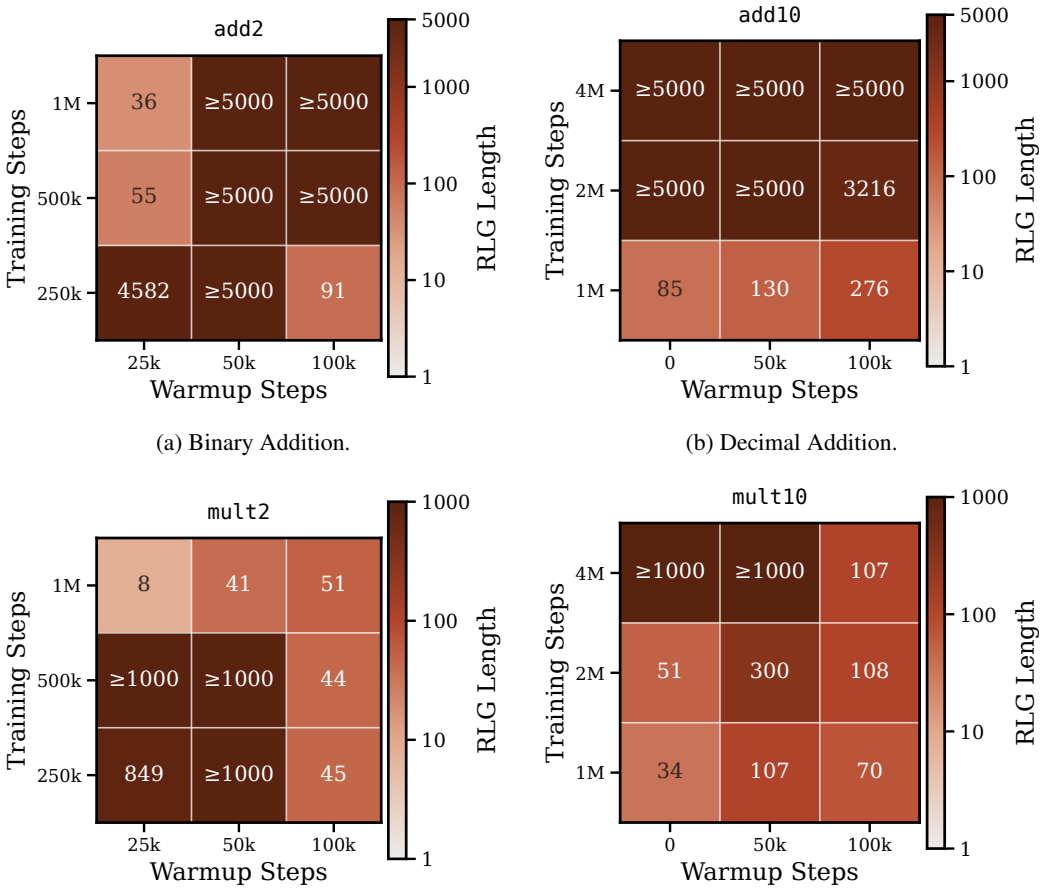
(d) Decimal Multiplication.

Figure 7: **DFST Uniform Length Generalization.** Each panel varies the update budget and warm-up duration. ULG requires exact action agreement along ten sampled expert trajectories; cap values denote successful finite probes. Figure 8 evaluates the same runs on structured inputs.

The addition curves separate random-input and structured-input generalization. The RNN already passes both caps at the first evaluated checkpoint. At 200k updates, DFST reaches ULG 5000 but RLG 2630; both measurements fall at 300k, to 1258 and 1062, before reaching both caps at 400k. GRU reaches ULG 5000 at 500k updates while RLG reaches only 526; RLG subsequently reaches 5000 at 900k. These trajectories show that additional optimization can improve behavior on structured inputs after random-input tests saturate, and that intermediate generalization need not improve monotonically. Figure 11 supplies the corresponding ULG curves.

All multiplication runs reach both caps by 400k updates. After first reaching an RLG cap, every later recorded checkpoint retains the cap. Across both tasks, the shared outcome supports state tracking with PTO supervision across multilinear, ungated recurrent, and gated recurrent controllers. Different hidden dimensions, initializations, seeds, and learning-rate schedules prevent a causal ranking of controller architectures; the histories establish successful learning for the selected configurations without quantifying success probabilities over random seeds.

The DFST multiplication trajectory uses a learning-rate floor of 10^{-9}, distinct from the 10^{-7} floor in the schedule sweep. Both GRU runs include a constant-rate phase between exponential warm-up and cooldown. The complete settings appear in Table 5.



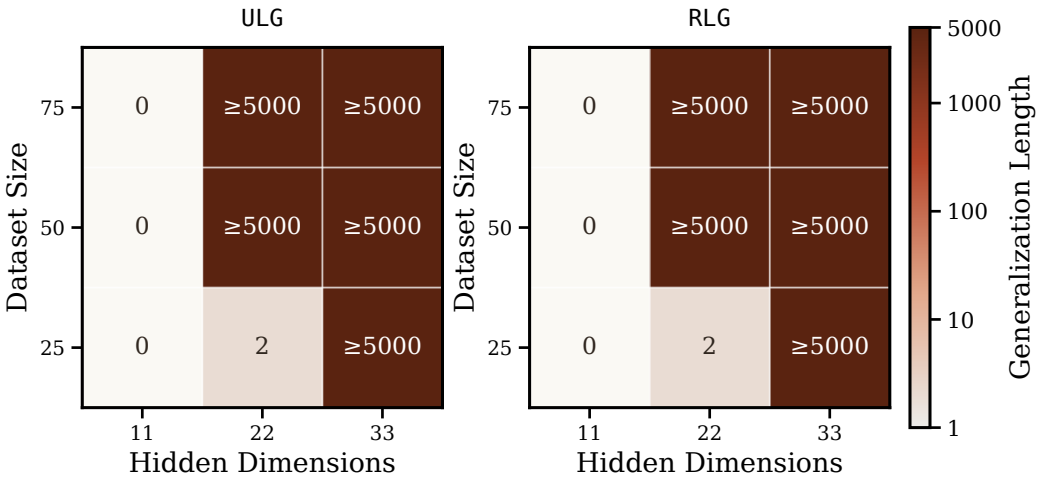
(a) Binary Addition.

(b) Decimal Addition.

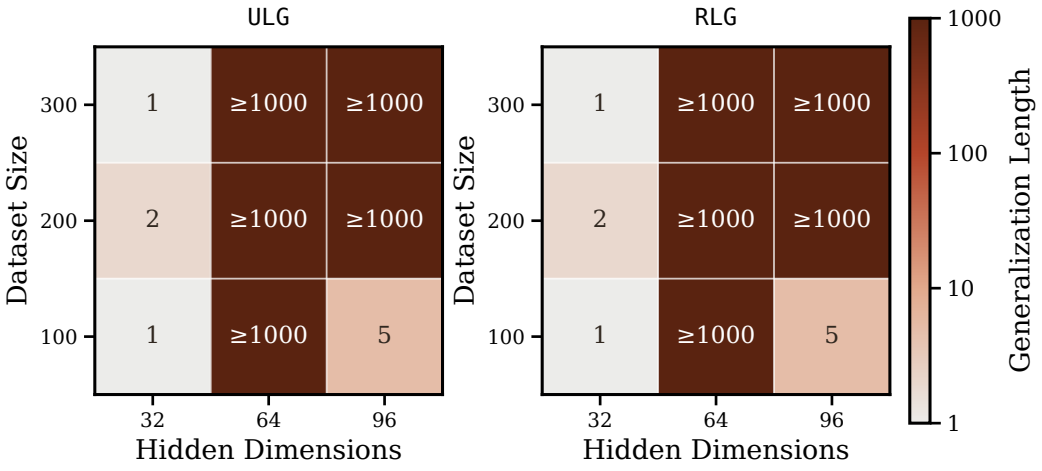
(c) Binary Multiplication.

(d) Decimal Multiplication.

Figure 8: **DFST Robust Length Generalization.** The training schedules match Figure 7. RLG supplements random-input testing with every repdigit cross-pair, four in binary and 100 in decimal. Cap values denote successful finite probes.

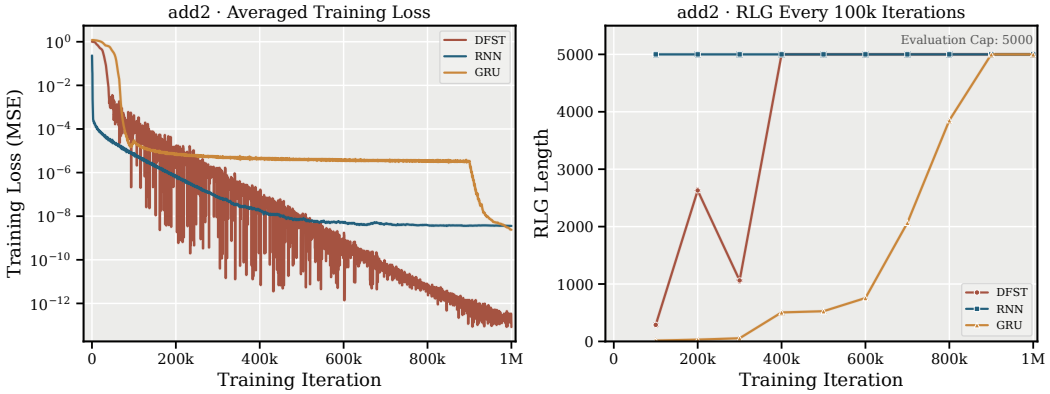


(a) Binary Addition.

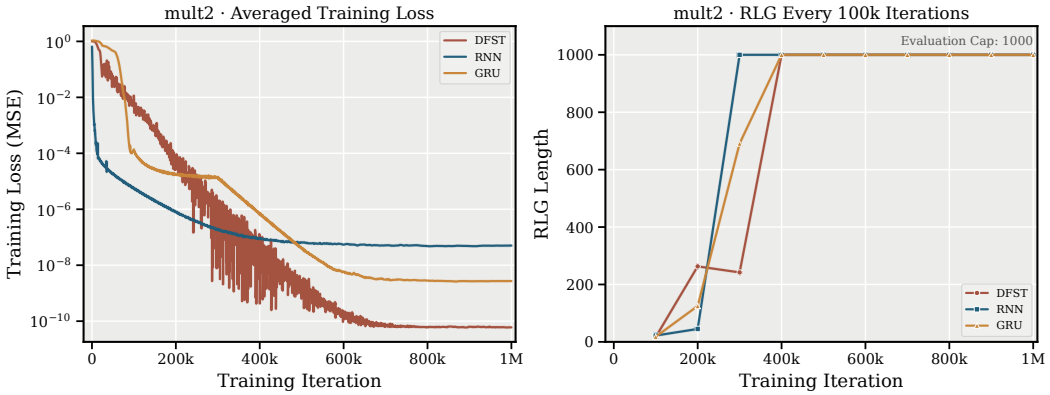


(b) Binary Multiplication.

Figure 9: **Initial Softmax DFST Generalization.** Each panel compares ULG and RLG across hidden dimensions and training-set sizes after 100k updates. The displayed results precede the annealing stage.

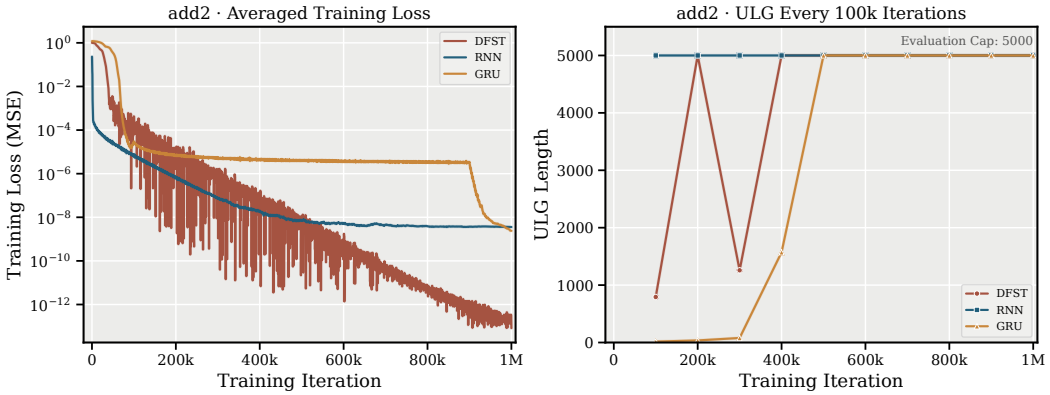


(a) Binary Addition: 5000-Digit Cap.

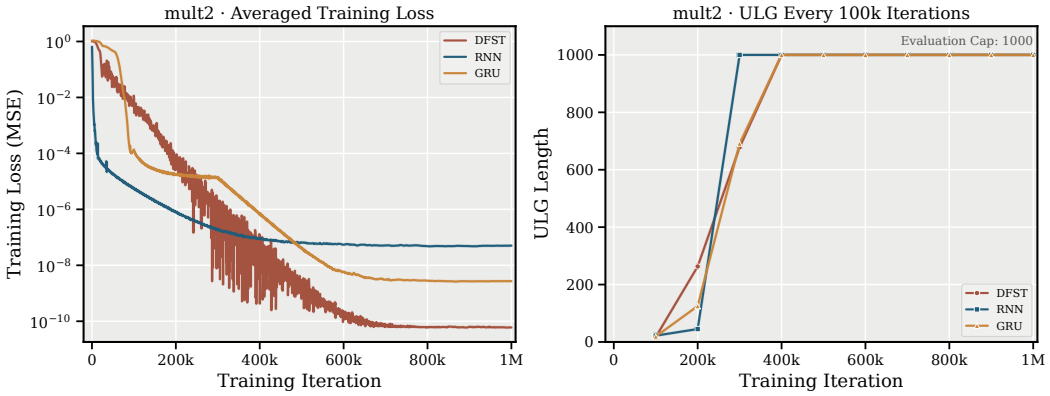


(b) Binary Multiplication: 1000-Digit Cap.

Figure 10: **Training Dynamics for Three State-Tracking Controllers.** Left panels show training MSE averaged over 1000 updates; right panels show RLG measured every 100k updates using all four binary repdigit pairs. Every final checkpoint reaches the task cap. Curves describe selected runs with different dimensions, seeds, initializations, and schedules (Table 5), rather than an isolated comparison of architectures.



(a) Binary Addition: 5000-Digit Cap.



(b) Binary Multiplication: 1000-Digit Cap.

Figure 11: **Uniform Generalization during Training.** The loss curves and checkpoints match Figure 10; the right panels show ULG. Comparing ULG and RLG distinguishes random-input success from robustness to repdigit inputs.

F Universal-Agent Experiments

The universal expert uses 15 states, a nine-symbol alphabet, and eight motion tokens. A single learned controller receives the encoded program through the environment; no controller weights change between simulated tasks.

F.1 Teleportation Runes

The controller observes only the symbol under the grid head and emits a symbol to write and a motion token. The environment applies the write before the motion. The alphabet contains $\{\lambda, 0, 1, \mathtt{L}, \mathtt{R}, \mathtt{S}, X, Y, Z\}$; motions comprise $\{L, R, U, D, S, X, Y, Z\}$, with S stopping execution. The three symbols X, Y, Z provide relocatable bookmarks. Writing a rune stores the symbol underneath the rune at the current cell. Rewriting the same rune restores the hidden symbol at the previous location and relocates the bookmark. A motion token X, Y, or Z jumps directly to the corresponding bookmark; writing a rune alone does not trigger a jump. Jumping to an unplaced rune leaves the head in place; valid simulations place each bookmark before use. Ordinary writes overwrite the current cell and remove any rune occupying that cell. The simulation relation must therefore track both visible grid contents and hidden symbols beneath runes.

During simulation, Z marks the simulated tape head, X identifies the program-table block for the current simulated state, and Y bookmarks the instruction currently undergoing decoding. These operations shorten transport between the program and tape while preserving a finite controller and an unbounded environment.

F.2 Program Encoding

For a binary Turing machine with s states, including the halting state, choose ternary width $\kappa = \max\{1, \lceil \log_3 s \rceil\}$. Encode state indices using $\mathtt{L}, \mathtt{R}, \mathtt{S}$ for ternary digits 0, 1, 2. Order program-table rows lexicographically by state index and read symbol, with blank preceding 0 and 1. Each row contains

$$[\, x,\ \lambda,\ \mathrm{code}(q),\ 0,\ \mathrm{code}(q'),\ 0,\ w,\ m\,],$$

where (q', w, m) denotes the Turing-machine transition on (q, x). The two delimiter zeros separate state fields. State codes use most-significant trits first. Thus the table contains $3s$ rows and $2\kappa + 6$ columns, with three consecutive rows per state. Unoccupied grid cells contain blanks. For example, with $s = 3$, a transition $(q, x) = (0, 1) \mapsto (2, 0, R)$ contributes the row $[1, \lambda, \mathtt{L}, 0, \mathtt{S}, 0, 0, \mathtt{R}]$. Initially, X covers the blank in column 1 of the first program row. The input word occupies columns starting at 1 of the row below the program table, and the universal head starts at the first input symbol.

The controller starts in state 0. Each simulated step first records the tape-head position with Z and jumps to X. A vertical search within the current-state block selects the row matching the read symbol. The controller then crosses the current-state code and decodes the next-state code one trit at a time: Y bookmarks the instruction field, while X advances through matching state-code columns using vertical searches. After locating the successor-state block, the controller returns X to column 1, reads the write symbol and motion through Y, and jumps to Z to execute the tape action. A simulated S action performs the specified write and stops the universal controller. Rune relocation restores covered program symbols, allowing repeated interpretation of the same table without changing controller weights. The metric-extraction equivalence result in Section H covers correctly encoded programs and nonempty binary input words. The annealed controller also admits the empty word; the two theorem domains remain distinct.

The simulation requires $O(|Q_M|\,|M[z]|)$ actions, where Q_M denotes the state set of M, including the halting state, and $|M[z]|$ counts the actions of M on z, including the stopping action. Each teleportation counts as one action.

F.3 Training and Evaluation Programs

Training samples 75, 150, or 225 PTOs with input lengths at most five from five programs: `parity`, `shift-left`, `shift-right`, `scaling-A`(5), and `scaling-B`(5). Static sampling includes at least one example from each training program and fills the remaining slots by random task and input

sampling. The implemented sampler permits empty words, although the metric-extraction equivalence theorem restricts the unbounded claim to nonempty words.

Evaluation includes all five training families and five held-out programs: bit inversion (`flip`), prefix parity (`cum-sum`), parity-dependent bit deletion (`bit-delete`), reversal, and copying. The last two programs require repeated scans over growing tape regions. The scaling tests set the program parameter to the candidate input length, so evaluation at 500 increases both input length and encoded machine size. `scaling-A`(s) executes s nonhalting steps followed by one stopping action; `scaling-B`(s) follows a parity-dependent walk through a finite chain of states. These families test new program sizes as well as longer inputs.

Table 7 summarizes the implemented programs. Alternatively, readers can inspect animated executions through the public controller-animation notebook or the animation command, for example `del animate tp:copy '1011'`.

Table 7: **Implemented Binary Turing Machines.** Examples start from 1011 in cells $0, \ldots, 3$, with the head at cell 0. Notation $v@j$ places word v starting at cell j; λ denotes a blank, and all unlisted cells remain blank. Scaling examples use parameter $s = 5$.

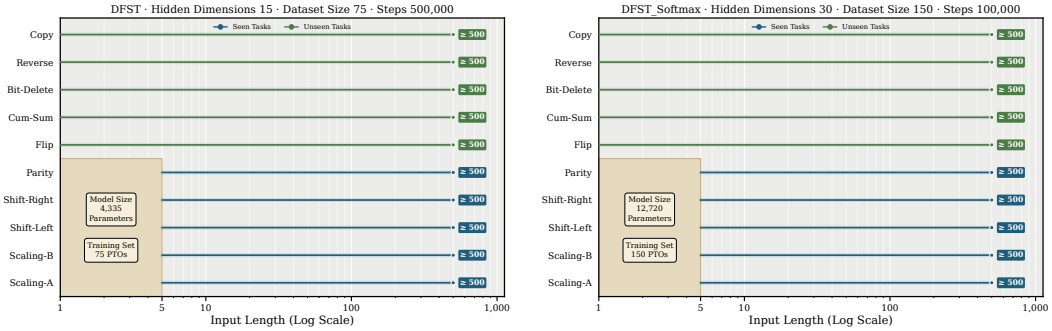
Program	Tape transformation	Final tape example
`parity`	Erases the input and writes the parity of the number of ones immediately after the original word.	$1@4$
`shift-left`	Moves the entire word one cell left, preserving bit order.	$1011@{-1}$
`shift-right`	Moves the entire word one cell right, preserving bit order.	$1011@1$
`scaling-A`(s)	Performs s state-indexed bit-flipping moves, then a symbol-preserving stopping action; see the rule below.	$00111@{-1}$
`scaling-B`(s)	Performs s parity-dependent rewrites, changing direction after encountering blanks; the final rewrite stops.	$11011@0$
`flip`	Replaces each bit b with $1 - b$ in place.	$0100@0$
`cum-sum`	Replaces each bit with the parity of the input prefix ending at that position.	$1101@0$
`bit-delete`	Retains zeros for even input length and ones for odd input length; replaces other bits with blanks without compacting the tape.	$\lambda 0 \lambda \lambda @0$
`reverse`	Reverses the word and relocates the result to cells $-(n-1), \ldots, 0$ for nonempty input of length n.	$1101@{-3}$
`copy`	Retains the input and appends a second copy after one blank cell.	$1011\lambda 1011@0$

For `scaling-A`(s), step $i = 0, \ldots, s-1$ maps 0 to 1 and moves left, or maps 1 to 0 and moves right. On a blank, step i writes $(i+1) \bmod 2$ and moves left for even i, right for odd i. For `scaling-B`(s), initialize parity $p = 0$ and direction right. Reading a bit b writes $p \oplus b$ and updates p to that value; reading a blank writes p and reverses the direction for the next step. Each nonfinal step moves in the direction held before the read; step s writes and stops.

F.4 Generalization across Training Configurations

Figure 13 collects every evaluated universal configuration across all ten tasks. All three ordinary-DFST runs with 500k updates reach the 500-symbol cap on every task. At 250k updates, the runs reach the cap on eight, eight, and nine tasks for $N = 75, 150, 225$, respectively. The remaining failures involve reversal, copying, or the second scaling family.

For Softmax DFSTs, two of nine configurations reach the cap on all ten tasks after 100k updates: $(d, N) = (30, 150)$ and $(45, 75)$. Several other configurations succeed on all eight fixed programs but fail on at least one scaling family. Thus generalization across fixed programs does not guarantee extrapolation in encoded machine size, and increasing dimension or dataset size does not uniformly improve the outcome.



(a) DFST: $d = 15$, $N = 75$, 500k Updates.

(b) Softmax DFST: $d = 30$, $N = 150$, 100k Updates.

Figure 12: **Generalization across Encoded Programs before Extraction.** Selected differentiably trained controllers reach the 500-symbol cap on all ten evaluation tasks. Training inputs contain at most five symbols. The scaling tests also increase the program parameter from 5 to 500.

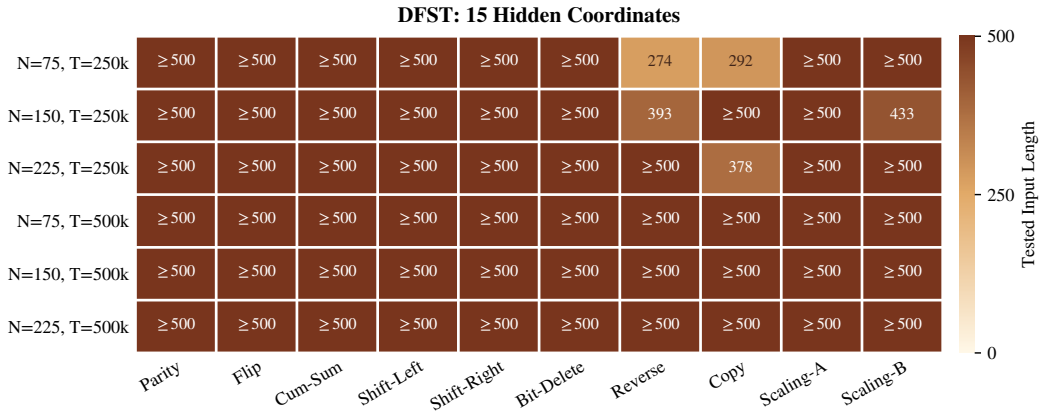
F.5 PTO Lengths at the Evaluation Caps

Operand length understates the execution horizon. Direct replay of the expert transition maps gives 50 070 010 PTO steps for adding two 5000-digit copies of the maximal digit and 15 012 991 steps for multiplying two 1000-digit copies, in both radix 2 and radix 10. These counts include exactly one stopping action. The maximal-digit examples belong to the evaluated repdigit family. The counts describe expert trajectories used by the cap probes; the counts do not constitute additional neural-model evaluations.

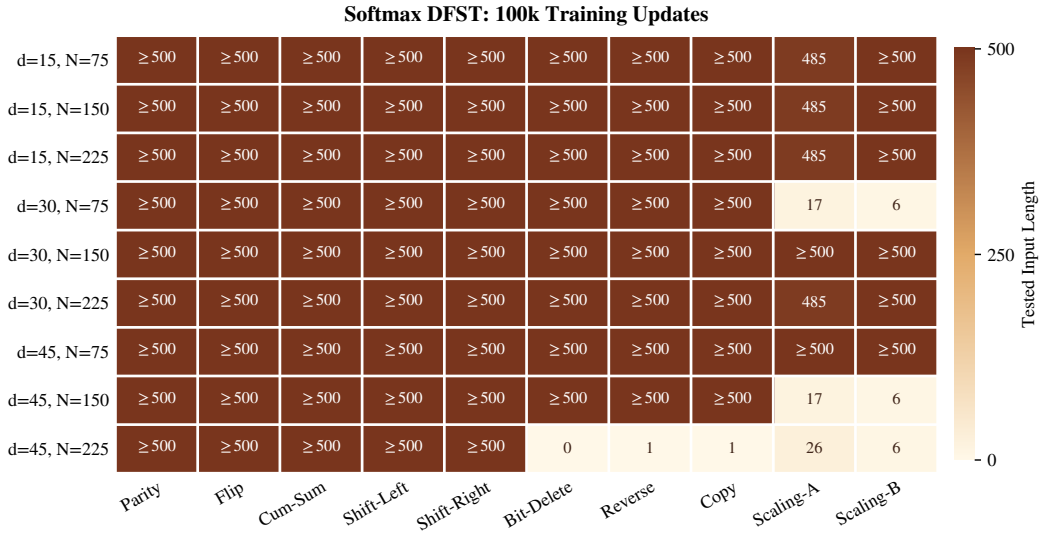
For universal simulation, replaying the ten deterministic evaluation inputs at length 500 for each task gives the maxima below. The longest sampled trajectory contains 15 316 822 steps, for copying. Thus a successful 500-symbol probe can require more than fifteen million consecutive action matches.

Table 8: Maximum Expert PTO Length among the 10 Seeded 500-Symbol Evaluation Inputs per Universal Task. Each count includes the final stopping action once.

Task	PTO steps	Task	PTO steps
`parity`	9295	`flip`	9295
`cum-sum`	9298	`bit-delete`	24 838
`shift-left`	41 806	`shift-right`	54 086
`reverse`	14 911 144	`copy`	15 316 822
`scaling-A`(500)	25 784	`scaling-B`(500)	427 803



(a) Ordinary DFST: Dataset Size and Update Budget.



(b) Softmax DFST: Hidden Dimension and Dataset Size.

Figure 13: **Complete Universal-Simulation Sweeps before Extraction.** Each cell reports the tested input length under exact PTO agreement. The notation ≥ 500 marks the evaluation cap. Training inputs contain at most five symbols; N counts PTOs, d denotes hidden dimension, and T counts training updates. Scaling tests increase the encoded machine size together with input length.

G Extracting Interpretable Finite-State Controllers

A trained recurrent controller already induces a finite-state machine under fixed finite-precision execution. Extraction seeks a compact description of the learned policy, suitable for interpretation and mathematical verification. We use two complementary constructions: metric extraction groups hidden states with similar geometry; annealing encourages a Softmax DFST to approach deterministic transitions. Neither construction uses expert hidden-state labels. The equivalence proofs in Section H concern the resulting discrete controllers. Transferring those proofs to the neural controllers requires the additional conditions stated below.

Executable notebooks accompany both metric extraction and annealing, with saved cases and instructions for analyzing another trained checkpoint.

G.1 Metric Extraction from DFST Trajectories

Autonomous Trajectory Collection. Fix a trained controller with hidden-state update f_x and decoded action g_x, where $x \in \Sigma$ and $g_x(h) \in \Sigma \times \mathcal{M}$. For a DFST, $f_x(h) = A[x]h$ and

$$g_x(h) = \big(\operatorname{argmax} B[x]h,\ \operatorname{argmax} C[x]h\big).$$

The readouts use the hidden state *before* the transition. From each sampled valid initial environment z, run the neural controller autonomously: the predicted write and movement determine the next observation. Record

$$(h_{t-1}, x_t, s_t, m_t, h_t), \qquad 1 \le t \le L(z).$$

Thus extraction observes the controller's actual policy rather than a teacher-forced continuation after an erroneous action. A stopping action contributes the final PTO triple exactly once. Nonfinite states and runs exceeding the collection budget trigger failure rather than a fictitious halt. The metric case studies use the unrescaled sequential DFST recurrence during collection; any transfer claim refers to the same arithmetic and execution convention.

Geometric Grouping and Deterministic Closure. For a finite trajectory collection, connect two nonterminal hidden-state occurrences whenever the Euclidean distance does not exceed ε. Take the connected components of this proximity graph. Connectedness permits chains of nearby points; a component need not have diameter at most ε. For each component q and observation x, inspect all recorded transitions. Conflicting write–movement pairs invalidate the threshold. Otherwise, merge the successor components whenever the same pair (q, x) reaches several successors, and repeat the consistency check and successor merging. Keep a distinguished halt state separate from nonterminal components; a required halt/nonhalt merger also invalidates the threshold. Every successful merger reduces the finite component count, so closure either terminates with a deterministic graph or detects a conflict. In the accepted graph $G(S, \varepsilon)$, an unobserved state–symbol pair remains undefined.

Threshold selection uses the complete extraction sample before measuring graph stability. Starting from a proposed ε, halve the threshold after a conflict and restart from the original observations. The implementation permits 65 positive-threshold attempts followed by a zero-threshold attempt; an unresolved conflict aborts extraction. A successful threshold supplies consistency with the collected trajectories, without certifying behavior beyond those trajectories.

Exact Quotient Reduction. Redirect stopping transitions to a single halt node, preserving the written symbol. Conventional halt self-loops only describe behavior after execution has stopped. Apply Hopcroft partition refinement to the resulting partial Mealy transducer, distinguishing final status, output labels, and definedness. Two states can share a quotient class only when every symbol has matching definedness and, for each defined transition, matching outputs and successor classes. The quotient therefore preserves all defined traces and all missing transitions. This reduction differs from completing unknown transitions or merging states merely because the observed samples fail to distinguish the states. The case studies additionally check the quotient map transition by transition and replay every extraction trajectory.

What Graph Stability Requires. At the accepted threshold, add trajectories one at a time and compare the rooted, input/output-labeled graphs up to state renaming. The comparison preserves the initial state, halt status, missing edges, and every transition label. Equal numbers of nodes and edges alone do not establish graph identity. In the five case studies, the final unchanged suffix begins at sample 4, 42, 143, 1,870, and 23 for `add2`, `mult2`, `add10`, `mult10`, and `universal`, respectively. Figure 14 displays the corresponding finite observations.

Task	Inputs	ε	Raw states	Raw edges	Quotient states
`add2`	100	0.5	23	44	22
`mult2`	100	0.125	70	133	61
`add10`	250	0.5	52	223	52
`mult10`	2,000	0.25	195	1,282	182
`universal`	100	0.5	49	78	44

Table 9: **Metric Extraction Case Studies.** Raw counts describe the final deterministic graph before quotient reduction and before conventional halt loops. State counts include the halt node. Arithmetic extraction samples use operands of at most 10 digits for addition and five digits for binary multiplication. Decimal multiplication combines 355 random inputs with operands of at most two digits and 1,645 protocol-covering teaching inputs with operands of at most five digits; these inputs support extraction and do not constitute additional gradient-training examples. Universal extraction uses length-five inputs across all ten evaluation task families.

Assumption H.1 requires substantially more: every finite enlargement of the sample by valid executions must preserve the same rooted labeled graph. For a nonhalting universal computation, the requirement extends to every finite execution prefix. A finite plateau provides evidence for this assumption, never a proof. Under the assumption, every valid neural trajectory follows a path already represented by the stable graph. Trace-preserving quotient reduction then transfers the neural policy to the compact controller, as formalized in Theorem 4.3. The assumption must refer to the deployed recurrence, rounding convention, and state normalization; changing those conventions requires a corresponding argument.

G.2 Annealing and Argmax Extraction from Softmax DFSTs

A Differentiable Probabilistic Parametrization. The model `DFST_Softmax` replaces unconstrained DFST tensors by column-stochastic tensors. For trainable logits $\Theta^A, \Theta^B, \Theta^C$ and θ^h, define

$$A[x]_{ij} = \frac{\exp(\Theta^A[x]_{ij})}{\sum_{k=1}^{d} \exp(\Theta^A[x]_{kj})}, \qquad h_{0,i} = \frac{\exp(\theta^h_i)}{\sum_{k=1}^{d} \exp(\theta^h_k)}, \tag{3}$$

$$B[x]_{sj} = \frac{\exp(\Theta^B[x]_{sj})}{\sum_{u\in\Sigma} \exp(\Theta^B[x]_{uj})}, \qquad C[x]_{mj} = \frac{\exp(\Theta^C[x]_{mj})}{\sum_{v\in\mathcal{M}} \exp(\Theta^C[x]_{vj})}. \tag{4}$$

The case studies use a fixed one-hot input embedding. In exact arithmetic, $h_t = A[x_t]h_{t-1}$ remains in the probability simplex, and $B[x_t]h_{t-1}$ and $C[x_t]h_{t-1}$ give distributions over writes and movements. Execution selects the largest coordinate of each readout. Softmax changes the parameterization, while the recurrence still admits the associative matrix scan used for DFST training. A deterministic FST corresponds to simplex vertices: one-hot initial state and one-hot columns of all three tensors.

Differentiable Concentration toward Deterministic Transitions. After ordinary PTO training, continue optimizing the same supervised objective with a concentration penalty. With $K = |\Sigma|$, define

$$\mathcal{R}(\theta) = \sum_{P\in\{A,B,C\}} \frac{1}{Kd} \sum_{x\in\Sigma} \sum_{j=1}^{d} \left(1 - \sum_i P[x]_{ij}^2\right) + 1 - \sum_i h_{0,i}^2. \tag{5}$$

Each summand vanishes at a simplex vertex and penalizes distributed probability mass. At update $k = 1, \ldots, N$, optimize

$$\mathcal{L}_{\mathrm{PTO}}(\theta) + \lambda_k \mathcal{R}(\theta), \qquad \lambda_k = \lambda_{\mathrm{start}} \left(\frac{\lambda_{\mathrm{end}}}{\lambda_{\mathrm{start}}}\right)^{(k-1)/(N-1)}. \tag{6}$$

All three case studies start from a 100,000-update checkpoint and use a fresh Adam optimizer at constant learning rate 10^{-3} for $N = 5,000$ additional updates, with $\lambda_{\mathrm{start}} = 10^{-5}$ and $\lambda_{\mathrm{end}} = 10^{5}$. The continuation retains the training data, supervised loss, clipping, and sampling settings. Thus annealing strengthens a differentiable penalty, rather than imposing a hard state assignment during optimization. The phrase *exponential annealing* refers to the regularization schedule, not to a softmax-temperature schedule.

Reading Out the Discrete Controller. For each hidden coordinate j and symbol x, set

$$\begin{aligned} \delta(j,x) &= \operatorname*{argmax}_i A[x]_{ij}, \quad & w(j,x) &= \operatorname*{argmax}_s B[x]_{sj}, \\ m(j,x) &= \operatorname*{argmax}_v C[x]_{vj}, & q_0 &= \operatorname*{argmax}_i h_{0,i}. \end{aligned} \tag{7}$$

Fix deterministic tie breaking. Retain states reachable from q_0, introduce the halt convention, and apply the same exact quotient reduction as above. The extracted transition reads the pre-transition coordinate j, matching the neural recurrence. Unlike metric extraction, argmax extraction defines an action for every symbol at each retained state. The final minimization yields 17 states for binary addition, 52 for binary multiplication, and 28 for universal simulation, including halt states. Argmax supplies a deterministic decoder of differentiably learned probabilities; the decoder itself does not supply gradients or an equivalence proof.

Uniform Leakage and Readout Margins. Fix a deterministic recurrent implementation, including rounding, and the effective readout tensors B, C with entries in $[0,1]$. Let $\widehat{T}$ denote the unminimized columnwise-argmax controller before adjoining the conventional halt node. Each state q corresponds to the one-hot vector e_q; motion S terminates execution. Fix tie breaking and the valid input domain $\mathcal{Z}$.

For each $z \in \mathcal{Z}$, run $\widehat{T}$ and record the pre-action states $q_t(z)$ and observations $x_{t+1}(z)$. Drive the implemented neural recurrence from the trained initial state with this same observation stream, obtaining $h_t(z)$. The index t includes every pre-action state through the stopping action, or every finite time for a nonhalting execution. Define the hidden-state leakage by

$$\ell_t(z) = \|h_t(z) - e_{q_t(z)}\|_1 = |h_{t,q_t(z)}(z) - 1| + \sum_{j \neq q_t(z)} |h_{t,j}(z)|. \tag{8}$$

This definition controls both off-target mass and drift of the intended coordinate, without assuming that rounded states remain exactly normalized.

For $P \in \{B, C\}$, write $a_P(q,x) = \operatorname{argmax}_i P[x]_{iq}$ for the prescribed write or motion. Over state–symbol pairs occurring on valid executions of $\widehat{T}$, define the minimum readout margin

$$\mu = \min_{(q,x),\, P \in \{B,C\}} \left(P[x]_{a_P(q,x),q} - \max_{b \neq a_P(q,x)} P[x]_{bq} \right). \tag{9}$$

The minimum ranges over finitely many columns. A certified superset of the reachable pairs supplies a conservative lower bound. Let $\widetilde{y}_t^P(z)$ denote the implemented readout, while $P[x_{t+1}(z)]h_t(z)$ denotes the exact real matrix-vector product of the stored tensor and hidden-state values.

Assumption G.1 (Uniformly Bounded Leakage)**.** There exist constants $\varepsilon, \rho \geq 0$, independent of input and execution length, with $\varepsilon + 2\rho < \mu$, such that every $z \in \mathcal{Z}$ and every pre-action time t satisfy

$$\ell_t(z) \leq \varepsilon, \qquad \|\widetilde{y}_t^P(z) - P[x_{t+1}(z)]h_t(z)\|_\infty \leq \rho \quad \text{for } P \in \{B, C\}. \tag{10}$$

The constant ρ bounds per-component readout rounding error; exact readout arithmetic permits $\rho = 0$.

Proposition G.2 (Faithfulness under Bounded Leakage)**.** *Under Assumption G.1, the neural controller and $\widehat{T}$ produce identical PTOs on all valid inputs, including simultaneous stopping and every finite prefix of a nonhalting execution. Any trace-preserving quotient inherits this equivalence.*

Proof. Fix z, t, put $q = q_t(z)$, $x = x_{t+1}(z)$, and $h = h_t(z)$, and consider either readout head P. Let $a = a_P(q,x)$ and let $b \neq a$ denote any competing action. Since the entries of $P[x]$ lie in $[0,1]$, the row difference satisfies $\|P[x]_{a:} - P[x]_{b:}\|_\infty \leq 1$. Therefore

$$\begin{aligned} (P[x]h)_a - (P[x]h)_b &= P[x]_{aq} - P[x]_{bq} + (P[x]_{a:} - P[x]_{b:})(h - e_q) \\ &\geq \mu - \|h - e_q\|_1 \geq \mu - \varepsilon. \end{aligned}$$

Readout rounding changes each score by at most ρ, so the implemented gap satisfies

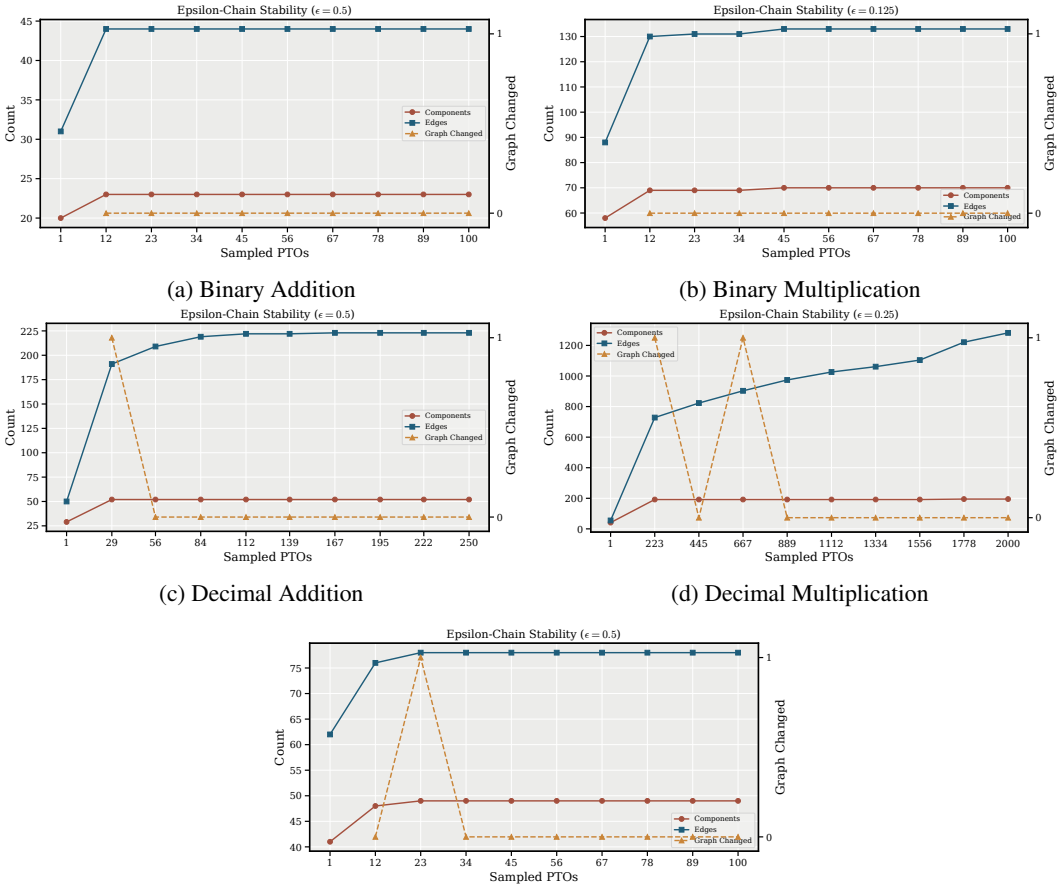
$$\widetilde{y}_{t,a}^P(z) - \widetilde{y}_{t,b}^P(z) \geq \mu - \varepsilon - 2\rho > 0.$$

Thus both implemented heads choose the prescribed actions on the supplied observation stream.

Now compare autonomous executions. Initially, both controllers receive the same grid, head position, and rune records. If all earlier actions agree, both environments supply $x_{t+1}(z)$, and the deterministic neural implementation reaches the same $h_t(z)$ as on the supplied stream. The margin argument gives the same write and motion, preserving the environment correspondence. A stopping action terminates both executions; otherwise induction continues. This proves agreement through every finite prefix and simultaneous stopping. Trace-preserving quotient reduction retains every such PTO. □

Sharp readout columns allow a larger tolerance for hidden-state leakage. For example, if every prescribed entry exceeds $1 - \beta$ and every competing entry remains below β, then $\mu > 1 - 2\beta$. Uniform leakage bounds concern the entire recurrent execution, whereas column sharpness concerns individual parameters. Exact one-hot tensors and initialization satisfy Assumption G.1 with $\varepsilon = \rho = 0$ whenever the implemented arithmetic preserves vertex operations.

Evidence and Proof Play Different Roles. The saved finite checks for metric binary addition, binary multiplication, decimal addition, and universal simulation, and for all three annealed controllers, each report 100 exact PTO matches on 100 inputs of length 100. Arithmetic checks use two length-100 operands; universal checks sample length-100 binary words and task instances. These checks compare the *extracted* controller against the expert. The neural length-generalization experiments in Section 3 provide separate evidence before extraction. Neither evidence source replaces the inductive proofs in Section H, and neither proves Assumptions G.1 and H.1. The eight proofs establish exact algorithmic learning for the extracted controllers; the stated graph-stability or bounded-leakage assumptions transfer the corresponding conclusion to the trained neural controllers.



(a) Binary Addition

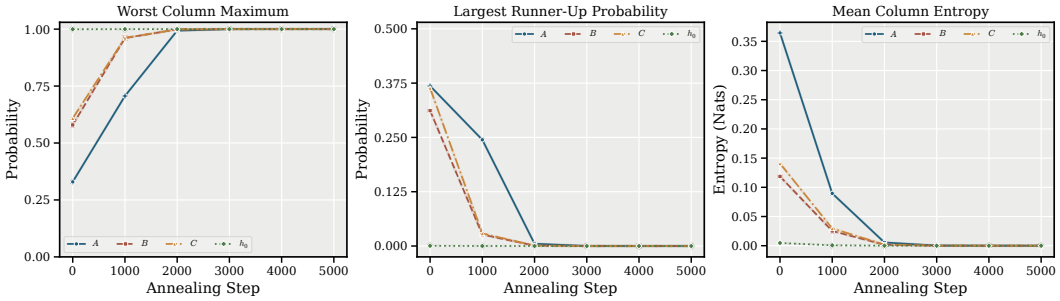
(b) Binary Multiplication

(c) Decimal Addition

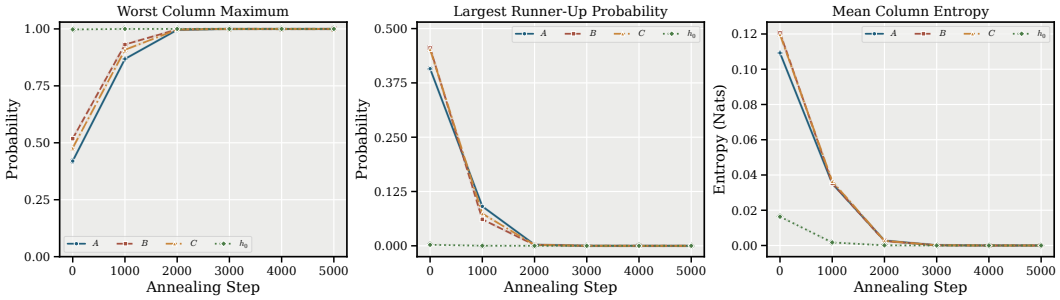
(d) Decimal Multiplication

(e) Universal Controller

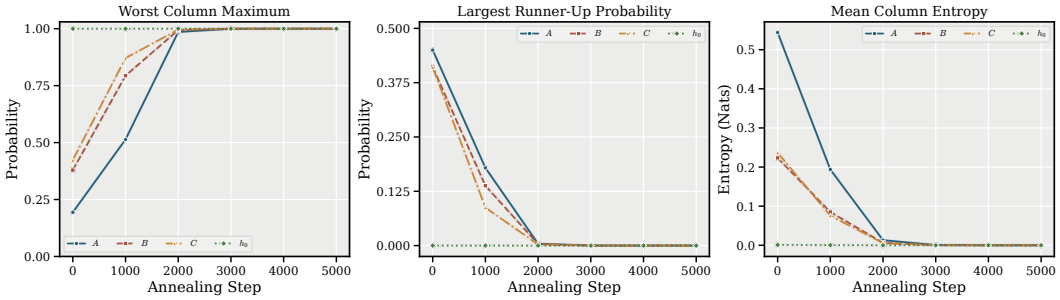
Figure 14: **Empirical Graph Stability under Metric Extraction.** The plots track components, observed edges, and changes as additional autonomous trajectories enter the extraction sample. The graph comparison also checks full rooted labeled structure independently of state numbering. Late plateaus support the geometric hypothesis on the sampled inputs; Assumption H.1 quantifies over all valid executions.



(a) Binary Addition



(b) Binary Multiplication



(c) Universal Controller

Figure 15: **Concentration during Differentiable Annealing.** Exponentially increasing the concentration penalty sharpens the transition and readout columns. At the final update, the smallest column maximum across A, B, C, h_0 exceeds $1 - 1.2 \cdot 10^{-7}$ for both arithmetic controllers and $1 - 2.4 \cdot 10^{-7}$ for the universal controller. These finite-precision measurements demonstrate near-deterministic parameters, but do not establish equivalence with the extracted FST for unbounded execution lengths.

H Proving Exact Learning

LLM Assistance. We used GPT-6 Astra to generate the controller-equivalence proofs and verified the arguments ourselves. Executable certificate checkers validate the finite transition obligations; the unbounded equivalence claims additionally rely on the mathematical invariants and induction arguments presented below. We take responsibility for the correctness of the final statements and proofs.

We separate three claims: equality of sampled PTOs, exact equivalence of an extracted controller, and transfer of that equivalence to the trained neural controller. The proofs below establish the second claim without a length bound. Transfer requires the graph-stability or bounded-leakage assumptions discussed in Section 4; finite experiments alone establish neither assumption.

The public proof bundles supply machine-readable transition maps and executable checks of the finite certificate obligations. The arguments below establish the unbounded claims independently of the accompanying code.

H.1 Why Finite Testing Cannot Certify General Equivalence

With fixed weights and deterministic arithmetic, D recurrent scalars stored in b bits induce at most 2^{bD} memory encodings. Restricting each scalar to a finite IEEE binary32 value gives $(2^{32} - 2^{24})^D$ encodings, counting signed zeros separately. Here $D = d$ for DFSTs, Softmax DFSTs, RNNs, and GRUs. This enormous FST remains difficult to inspect; ordinary FST equivalence on arbitrary read strings does not resolve equivalence under an unbounded environment.

Proof of Lemma 4.1. Suppose an algorithm computes the claimed horizon. Given a Turing machine N and input w, construct two grid agents that simulate exactly $|z|$ steps of $N(w)$ on input z, using identical visible actions throughout the simulation. If the simulation detects halting, the agents emit different final actions; otherwise both agents stop with the same PTO. Every such execution terminates. The agents agree on every input precisely when $N(w)$ never halts. A computed horizon would reduce this universal equivalence claim to finitely many terminating comparisons, deciding nonhalting and therefore halting. This contradiction proves the claim. The obstruction concerns controllers interacting with an unbounded environment; ordinary finite-state transducer equivalence on unrestricted read strings remains decidable. □

H.2 Transferring an Extracted Certificate to a Trained Controller

Let $G(S, \varepsilon)$ denote the deterministic, trace-consistent graph extracted from valid neural executions S at threshold ε, including finite prefixes of nonhalting executions.

Assumption H.1 (Graph Stability)**.** For some $\varepsilon > 0$ and finite sample S, every finite extension S' by valid neural execution traces or prefixes yields a successful extraction and a graph $G(S', \varepsilon)$ isomorphic to $G(S, \varepsilon)$, preserving the root, read symbols, output labels, and stopping actions.

Proof of Theorem 4.3. For metric extraction, fix a valid input and any finite execution prefix of the trained controller. Extend the extraction data by that prefix. Trace consistency embeds every observed transition into the resulting graph. Graph stability identifies the extended rooted, input/output-labeled graph with the original graph, including the initial node and stopping convention. Thus the original graph reproduces every action of the prefix. Induction synchronizes the controller and graph: equal grids and head positions provide equal read symbols; equal output labels produce equal writes and motions and preserve the next configuration. Rune records, when present, evolve identically under the same actions. Since the argument permits every prefix length, the complete finite PTOs agree, or every finite prefix agrees for a nonterminating execution. Any subsequent merging that preserves all defined traces preserves this conclusion. Composing this equality with an extracted-controller certificate proves equivalence to the expert. Equality of node or edge counts alone would not supply the labeled-graph identity used in this argument. For Softmax DFSTs, Proposition G.2 establishes equality of every finite PTO prefix under Assumption G.1. Trace-preserving reduction and composition with the expert-equivalence certificate give the same conclusion. □

H.3 A Common Simulation Principle

Write $\mathcal{E}$ for an expert and $\mathcal{C}$ for an extracted controller. A transition $\delta(q, x) = (q', w, M)$ reads x, writes w, performs motion M, and enters state q'. Motion S writes and stops without moving. PTOs contain read/write/motion triples and count the final stopping action once; halt loops contribute no additional observations. Arithmetic inputs occupy row 0 from column 0, with all other cells blank and the head initially at $(0, 0)$. Arithmetic coordinates increase rightward and downward. Operand words may have unequal positive lengths and leading zeros.

A *phase* records the expert control state together with finite geometric or arithmetic context. A phase need not correspond to a distinct machine state. The following lemma explains why a finite certificate can establish equality of arbitrarily long executions.

Lemma H.2 (Phase Simulation)**.** *Suppose a protocol covers every valid expert execution. Assign expert state g_i and a nonempty controller-state set Q_i to each phase i. Assume the initial states belong to the initial phase relation. For each permitted edge $i \xrightarrow{x} j$ and every $q \in Q_i$, suppose*

$$\delta_{\mathcal{E}}(g_i, x) = (g_j, w, M), \qquad \delta_{\mathcal{C}}(q, x) = (q', w, M), \qquad q' \in Q_j. \tag{11}$$

Assume stopping occurs exactly on edges to terminal phases. Then the two controllers produce identical PTOs on the valid input domain, with identical environment configurations at every step.

Proof. Relate configurations with the same grid, head, and, where applicable, rune records, and control states g_i and $q \in Q_i$. Initially this relation holds. Protocol coverage provides an allowed read symbol; equal configurations provide the same symbol to both controllers. Equation (11) gives the same write and motion and relates the successor states. The deterministic environment update preserves the relation. Induction proves equality of every finite prefix. The stopping condition gives simultaneous termination, or equal infinite PTOs. No undefined transition can occur along a valid execution. □

The tables below give the necessary transition maps on valid executions. An entry $x : (j, w, M)$ specifies a successor *phase* j, with expert successor g_j and controller successor in Q_j; ordinary transition tables instead list actual successor states. Multivalued Q_i capture different internal realizations of the same phase. Each listed branch applies to every state in the corresponding set. Undefined or unlisted branches require no completion when the coverage proof excludes those branches. Local identities can therefore be checked by finite substitution, while the geometric invariants supply the unbounded claim.

Task	Extraction	States	Phases	Related pairs	Local checks
`add2`	Metric	22	27	28	50
	Annealing	17	27	30	52
`mult2`	Metric	61	67	67	130
	Annealing	52	94	109	201
`add10`	Metric	52	59	60	266
`mult10`	Metric	182	207	208	1,509
`universal`	Metric	44	92	96	160
	Annealing	28	91	93	162

State counts include the designated halt state. All eight finite certificates satisfy the local obligations. The following arguments establish coverage for every input in each stated domain.

H.4 Binary Addition: Two Certificates, One Arithmetic Invariant

Let $D = \{0, 1\}$ and consider $a + b$ with $a \in D^m$, $b \in D^n$, $m, n \geq 1$. Write $A = \mathrm{val}_2(a)$ and $B = \mathrm{val}_2(b)$. The expert starts in state 0 and halts in state 10. The metric and annealed controllers start in state 0 and halt in states 21 and 16, respectively.

Theorem H.3 (Exact Binary Addition)**.** *Both extracted controllers produce the expert PTO on every valid binary-addition input, with identical grids and head positions at every step.*

H.4.1 Shared Expert Behavior and Observation Protocol

Accumulator and Marker Invariant Write $B = \sum_{j=0}^{n-1} b_j 2^j$, with b_0 the rightmost digit. The expert scans to the separator at column m, steps left and down, writes a working $+$ below column $m-1$, and enters state 8 above the marker.

Lemma H.4 (Round Invariant)**.** *After processing k right-operand digits, $0 \le k \le n$, the expert enters state 8 at $(p_k, 0)$, where $p_k = m - 1 - k$. At this point:*

- *Row 1 contains exactly one nonblank symbol: $a +$ at column p_k.*
- *Row 0 contains a contiguous accumulator on $[\ell_k, m-1]$, with value $C_k = A + \sum_{j<k} b_j 2^j$ and left edge $\ell_k \le p_k + 1$. Column $m-1$ carries the units digit.*
- *The separator remains at column m. The unprocessed prefix of b occupies columns $m+1, \ldots, m+n-k$. All remaining cells contain blanks.*

Proof. Initialization gives $k = 0$, $\ell_0 = 0$, $C_0 = A$, and $p_0 = m - 1$. State 8 preserves the digit at p_k, or writes 0 on a blank. A blank at p_k forces $\ell_k = p_k + 1$, so zero extension preserves contiguity.

For $k < n$, the right scan reaches the blank after the unprocessed prefix, steps left to digit b_k at $r_k = m + n - k$, erases b_k, and descends. The expert transports b_k left along row 1 to the marker. The distance satisfies

$$r_k - p_k = n + 1 \ge 2. \tag{12}$$

Consequently, the first two transport steps read blanks. At the marker, the expert ascends and adds b_k to the accumulator digit e at column p_k. If $e + b_k < 2$, the expert writes $e + b_k$ and descends. Otherwise, the expert writes $e + b_k - 2$, moves left, and propagates one carry. Each digit 1 changes to 0; the first smaller digit increases by one, or a blank changes to 1. The update adds exactly $b_k 2^k$ and preserves contiguity.

The expert returns along row 1, erases the marker, steps left, writes the next marker at $p_{k+1} = p_k - 1$, and ascends into state 8. The earlier zero extension gives $\ell_{k+1} \le p_k = p_{k+1} + 1$. The accumulator now has value C_{k+1}, establishing the next round. □

Lemma H.5 (Termination and Final Configuration)**.** *The expert terminates with only the padded base-2 representation of $A + B$ on row 0, ending at column $m - 1$. The final head stops at $(m - n - 1, 0)$. Writing $\ell_2(s)$ for the binary digit count, with $\ell_2(0) = 1$, the output width equals $L = \max\{m, n + 1, \ell_2(A + B)\}$.*

Proof. Each round consumes one of the n right-operand digits. Every scan crosses a finite interval, and every carry reaches a digit below 1 or a blank. After round n, state 8 ensures a digit at p_n. The next right scan returns to the separator. State 3 erases the separator and descends. State 9 walks left, erases the final marker, ascends, preserves the accumulator digit, and stops. The invariant gives $C_n = A + B$, and cleanup removes all work symbols. The original m columns persist, and marker extension reaches p_n, ensuring at least $n + 1$ digits. Every additional carry extension contributes a digit required by the current numerical value. Partial sums never decrease, so the final width equals L. □

The following phases refine the expert states by geometric context. The protocol covers every valid execution and may also admit additional observation sequences. Write $d \in \{0, 1\}$. An edge $x : j$ reads x and advances to phase j; $d : j$ permits either digit. Expert state g_i supplies the write and motion from Section H.4.2. A dash denotes a terminal phase with no outgoing transition.

i	Phase meaning	g_i	Allowed reading : next phase
0	First input digit	0	$d:1$
1	Seek the separator	0	$d:1$ $+:2$
2	Initial descent	1	$d:3$
3	Place the marker	1	$\lambda:4$
4	Ensure an accumulator digit	8	$d:6$ $\lambda:5$
5	Digit after zero extension	2	$d:6$
6	Scan toward separator	2	$d:6$ $+:7$
7	First cell after separator	2	$d:8$ $\lambda:22$
8	Scan remaining right operand	2	$d:8$ $\lambda:9$
9	Read rightmost factor digit	3	$0:10$ $1:14$
10	Transport zero: first blank	4	$\lambda:11$
11	Transport zero: second blank	4	$\lambda:12$
12	Transport zero: walk left	4	$\lambda:12$ $+:13$
13	Add zero above marker	4	$d:21$
14	Transport one: first blank	5	$\lambda:15$
15	Transport one: second blank	5	$\lambda:16$
16	Transport one: walk left	5	$\lambda:16$ $+:17$
17	Add one above marker	5	$0:21$ $1:18$
18	Propagate a carry	6	$1:18$ $0:19$ $\lambda:19$
19	First return cell after carry	7	$\lambda:20$
20	Return to the marker	7	$\lambda:20$ $+:3$
21	At marker, without carry	7	$+:3$
22	Erase the input separator	3	$+:23$
23	First cleanup cell	9	$\lambda:24$
24	Find and erase final marker	9	$\lambda:24$ $+:25$
25	Preserve result and stop	9	$d:26$
26	Halted	10	–

Lemma H.6 (Protocol Coverage)**.** *Every execution of $\mathcal{E}$ from a valid input follows this protocol, starting at phase* 0 *and ending at phase* 26*.*

Proof. The nonempty first operand establishes phases 0–3. Lemma H.4 places a digit immediately after a zero extension in phase 4, establishing phase 5. The right scan meets the separator before any blank. To the separator's right, a first blank means the next left step reads the separator (phase 22); after any digit, that step reads a digit (phase 9). Equation (12) guarantees two initial blank readings while transporting either bit. The cell above the marker contains a digit. A carry ends strictly left of the marker, so the first return cell contains a blank. A noncarry addition returns directly to the marker. Erasing and shifting the marker reaches a blank. Finally, cleanup starts below the separator on a blank, reaches the marker and then a digit. These facts justify every branch of the table. Induction over the steps and rounds proves coverage; Lemma H.5 supplies the final phase. □

H.4.2 Metric-Extraction Certificate

Phase Relation and Local Checks For phase i, write Q_i for the controller-state set in the following table. The relation pairs an expert configuration in phase i with a controller configuration having the *same grid and head* and a state in Q_i.

Phase i	Meaning	Expert state g_i	Controller states Q_i
0	First input digit	0	$\{0\}$
1	Seek the separator	0	$\{1\}$
2	Initial descent	1	$\{2\}$
3	Place the marker	1	$\{3\}$
4	Ensure an accumulator digit	8	$\{4\}$
5	Digit after zero extension	2	$\{15\}$
6	Scan toward separator	2	$\{5\}$
7	First cell after separator	2	$\{6\}$
8	Scan remaining right operand	2	$\{6\}$
9	Read rightmost factor digit	3	$\{7\}$
10	Transport zero: first blank	4	$\{12\}$
11	Transport zero: second blank	4	$\{13\}$
12	Transport zero: walk left	4	$\{13\}$
13	Add zero above marker	4	$\{14\}$
14	Transport one: first blank	5	$\{8\}$
15	Transport one: second blank	5	$\{9\}$
16	Transport one: walk left	5	$\{9\}$
17	Add one above marker	5	$\{10\}$
18	Propagate a carry	6	$\{18\}$
19	First return cell after carry	7	$\{19, 20\}$
20	Return to the marker	7	$\{20\}$
21	At marker, without carry	7	$\{11\}$
22	Erase the input separator	3	$\{7\}$
23	First cleanup cell	9	$\{16\}$
24	Find and erase final marker	9	$\{16\}$
25	Preserve result and stop	9	$\{17\}$
26	Halted	10	$\{21\}$

Local Transition Identity. For every listed $q \in Q_i$ and every allowed edge $i \xrightarrow{x} j$ from the protocol, the transition maps satisfy

$$\delta_{\mathcal{E}}(g_i, x) = (g_j, w, M), \qquad \delta_{\mathcal{C}}(q, x) = (q', w, M), \qquad q' \in Q_j. \tag{13}$$

Every required controller entry exists. A transition reaches the halt state exactly when $j = 26$, with motion S. The certificate contains **28 phase–state pairs** (including the terminal pair) and **50 local checks**. The checks cover 43 distinct nonterminal controller entries. Section H.4.2 provides the tables for direct inspection; direct substitution checks every instance of (13).

Transition Tables Each entry lists $(\text{next state}, \text{written symbol}, \text{motion})$. The symbol λ means blank; a dash means an undefined transition. State numbers identify the extracted controllers throughout each certificate.

Binary-Addition Expert. Initial state 0; halt 10.

State	λ	$+$	0	1
0	$(10, \lambda, \mathsf{S})$	$(1, +, \mathsf{L})$	$(0, 0, \mathsf{R})$	$(0, 1, \mathsf{R})$
1	$(8, +, \mathsf{U})$	$(10, +, \mathsf{S})$	$(1, 0, \mathsf{D})$	$(1, 1, \mathsf{D})$
2	$(3, \lambda, \mathsf{L})$	$(2, +, \mathsf{R})$	$(2, 0, \mathsf{R})$	$(2, 1, \mathsf{R})$
3	$(10, \lambda, \mathsf{S})$	$(9, \lambda, \mathsf{D})$	$(4, \lambda, \mathsf{D})$	$(5, \lambda, \mathsf{D})$
4	$(4, \lambda, \mathsf{L})$	$(4, +, \mathsf{U})$	$(7, 0, \mathsf{D})$	$(7, 1, \mathsf{D})$
5	$(5, \lambda, \mathsf{L})$	$(5, +, \mathsf{U})$	$(7, 1, \mathsf{D})$	$(6, 0, \mathsf{L})$
6	$(7, 1, \mathsf{D})$	$(10, +, \mathsf{S})$	$(7, 1, \mathsf{D})$	$(6, 0, \mathsf{L})$
7	$(7, \lambda, \mathsf{R})$	$(1, \lambda, \mathsf{L})$	$(10, 0, \mathsf{S})$	$(10, 1, \mathsf{S})$
8	$(2, 0, \mathsf{R})$	$(10, +, \mathsf{S})$	$(2, 0, \mathsf{R})$	$(2, 1, \mathsf{R})$
9	$(9, \lambda, \mathsf{L})$	$(9, \lambda, \mathsf{U})$	$(10, 0, \mathsf{S})$	$(10, 1, \mathsf{S})$
10	$(10, \lambda, \mathsf{S})$	$(10, +, \mathsf{S})$	$(10, 0, \mathsf{S})$	$(10, 1, \mathsf{S})$

Metric-Extracted Controller. Initial state 0; halt 21.

State	λ	$+$	0	1
0	–	–	$(1, 0, \mathsf{R})$	$(1, 1, \mathsf{R})$
1	–	$(2, +, \mathsf{L})$	$(1, 0, \mathsf{R})$	$(1, 1, \mathsf{R})$
2	–	–	$(3, 0, \mathsf{D})$	$(3, 1, \mathsf{D})$
3	$(4, +, \mathsf{U})$	–	–	–
4	$(15, 0, \mathsf{R})$	–	$(5, 0, \mathsf{R})$	$(5, 1, \mathsf{R})$
5	–	$(6, +, \mathsf{R})$	$(5, 0, \mathsf{R})$	$(5, 1, \mathsf{R})$
6	$(7, \lambda, \mathsf{L})$	–	$(6, 0, \mathsf{R})$	$(6, 1, \mathsf{R})$
7	–	$(16, \lambda, \mathsf{D})$	$(12, \lambda, \mathsf{D})$	$(8, \lambda, \mathsf{D})$
8	$(9, \lambda, \mathsf{L})$	–	–	–
9	$(9, \lambda, \mathsf{L})$	$(10, +, \mathsf{U})$	–	–
10	–	–	$(11, 1, \mathsf{D})$	$(18, 0, \mathsf{L})$
11	–	$(3, \lambda, \mathsf{L})$	–	–
12	$(13, \lambda, \mathsf{L})$	–	–	–
13	$(13, \lambda, \mathsf{L})$	$(14, +, \mathsf{U})$	–	–
14	–	–	$(11, 0, \mathsf{D})$	$(11, 1, \mathsf{D})$
15	–	–	$(5, 0, \mathsf{R})$	$(5, 1, \mathsf{R})$
16	$(16, \lambda, \mathsf{L})$	$(17, \lambda, \mathsf{U})$	–	–
17	–	–	$(21, 0, \mathsf{S})$	$(21, 1, \mathsf{S})$
18	$(20, 1, \mathsf{D})$	–	$(19, 1, \mathsf{D})$	$(18, 0, \mathsf{L})$
19	$(20, \lambda, \mathsf{R})$	–	–	–
20	$(20, \lambda, \mathsf{R})$	$(3, \lambda, \mathsf{L})$	–	–
21	$(21, \lambda, \mathsf{S})$	$(21, +, \mathsf{S})$	$(21, 0, \mathsf{S})$	$(21, 1, \mathsf{S})$

H.4.3 Annealing Certificate

Phase Relation and Local Checks For phase i, write Q_i for the controller-state set in the following table. The relation pairs an expert configuration in phase i with a controller configuration having the *same grid and head* and a state in Q_i.

Phase i	Meaning	Expert state g_i	Controller states Q_i
0	First input digit	0	$\{0\}$
1	Seek the separator	0	$\{0\}$
2	Initial descent	1	$\{7\}$
3	Place the marker	1	$\{3, 9\}$
4	Ensure an accumulator digit	8	$\{6\}$
5	Digit after zero extension	2	$\{6\}$
6	Scan toward separator	2	$\{4\}$
7	First cell after separator	2	$\{8\}$
8	Scan remaining right operand	2	$\{4\}$
9	Read rightmost factor digit	3	$\{13\}$
10	Transport zero: first blank	4	$\{2\}$
11	Transport zero: second blank	4	$\{2\}$
12	Transport zero: walk left	4	$\{2\}$
13	Add zero above marker	4	$\{7\}$
14	Transport one: first blank	5	$\{4\}$
15	Transport one: second blank	5	$\{13\}$
16	Transport one: walk left	5	$\{15\}$
17	Add one above marker	5	$\{1\}$
18	Propagate a carry	6	$\{1\}$
19	First return cell after carry	7	$\{5\}$
20	Return to the marker	7	$\{5\}$
21	At marker, without carry	7	$\{3, 5, 9\}$
22	Erase the input separator	3	$\{12\}$
23	First cleanup cell	9	$\{11\}$
24	Find and erase final marker	9	$\{14\}$
25	Preserve result and stop	9	$\{10\}$
26	Halted	10	$\{16\}$

Local Transition Identity. For every listed $q \in Q_i$ and every allowed edge $i \xrightarrow{x} j$ from the protocol, the transition maps satisfy

$$\delta_{\mathcal{E}}(g_i, x) = (g_j, w, M), \qquad \delta_{\mathcal{C}}(q, x) = (q', w, M), \qquad q' \in Q_j. \tag{14}$$

Every required controller entry exists. A transition reaches the halt state exactly when $j = 26$, with motion S. The certificate contains **30 phase–state pairs** (including the terminal pair) and **52 local checks**. The checks cover 37 distinct nonterminal controller entries. Section H.4.3 provides the tables for direct inspection; direct substitution checks every instance of (14).

Transition Tables Each entry lists (next state, written symbol, motion). The symbol λ means blank; a dash means an undefined transition. State numbers identify the extracted controllers throughout each certificate.

Annealed Controller. Initial state 0; halt 16.

State	λ	$+$	0	1
0	$(13, \lambda, \mathsf{L})$	$(7, +, \mathsf{L})$	$(0, 0, \mathsf{R})$	$(0, 1, \mathsf{R})$
1	$(5, 1, \mathsf{D})$	$(3, \lambda, \mathsf{L})$	$(5, 1, \mathsf{D})$	$(1, 0, \mathsf{L})$
2	$(2, \lambda, \mathsf{L})$	$(7, +, \mathsf{U})$	$(5, \lambda, \mathsf{D})$	$(4, \lambda, \mathsf{D})$
3	$(6, +, \mathsf{U})$	$(3, \lambda, \mathsf{L})$	$(5, 1, \mathsf{D})$	$(1, 0, \mathsf{L})$
4	$(13, \lambda, \mathsf{L})$	$(8, +, \mathsf{R})$	$(4, 0, \mathsf{R})$	$(4, 1, \mathsf{R})$
5	$(5, \lambda, \mathsf{R})$	$(9, \lambda, \mathsf{L})$	$(4, 0, \mathsf{R})$	$(4, 1, \mathsf{R})$
6	$(6, 0, \mathsf{R})$	$(7, +, \mathsf{L})$	$(4, 0, \mathsf{R})$	$(4, 1, \mathsf{R})$
7	$(6, +, \mathsf{U})$	$(3, \lambda, \mathsf{L})$	$(9, 0, \mathsf{D})$	$(3, 1, \mathsf{D})$
8	$(12, \lambda, \mathsf{L})$	$(8, +, \mathsf{R})$	$(4, 0, \mathsf{R})$	$(4, 1, \mathsf{R})$
9	$(6, +, \mathsf{U})$	$(9, \lambda, \mathsf{L})$	$(4, 0, \mathsf{R})$	$(8, 1, \mathsf{R})$
10	$(14, +, \mathsf{U})$	$(10, \lambda, \mathsf{D})$	$(16, 0, \mathsf{S})$	$(16, 1, \mathsf{S})$
11	$(14, \lambda, \mathsf{L})$	$(10, \lambda, \mathsf{D})$	$(16, 0, \mathsf{S})$	$(16, 1, \mathsf{S})$
12	$(15, \lambda, \mathsf{L})$	$(11, \lambda, \mathsf{D})$	$(4, \lambda, \mathsf{D})$	$(4, \lambda, \mathsf{D})$
13	$(15, \lambda, \mathsf{L})$	$(11, \lambda, \mathsf{D})$	$(2, \lambda, \mathsf{D})$	$(4, \lambda, \mathsf{D})$
14	$(14, \lambda, \mathsf{L})$	$(10, \lambda, \mathsf{U})$	$(4, 0, \mathsf{R})$	$(4, 1, \mathsf{R})$
15	$(15, \lambda, \mathsf{L})$	$(1, +, \mathsf{U})$	$(4, 0, \mathsf{R})$	$(4, 1, \mathsf{R})$
16	$(16, \lambda, \mathsf{S})$	$(16, +, \mathsf{S})$	$(16, 0, \mathsf{S})$	$(16, 1, \mathsf{S})$

Proof of Theorem H.3. The common round invariant proves protocol coverage and termination. The two state relations separately satisfy (11): the metric certificate contains 50 local identities, and the annealing certificate contains 52. Both relations contain the initial pair and terminate only after the common final S action. Apply Lemma H.2 to each relation. The final grid contains the padded representation of $A + B$ specified by Lemma H.5. □

H.5 Binary Multiplication: Shared Products and Refined Protocols

Consider $a\,x\,b$ with $a \in \{0,1\}^m$, $b \in \{0,1\}^n$, $m, n \geq 1$, where x denotes the multiplication separator. Write $A = \mathrm{val}_2(a)$ and $B = \mathrm{val}_2(b)$. The expert has initial state 0 and halt state 31. Both extracted controllers start in state 0; metric extraction yields halt state 60, and annealing yields halt state 51.

Theorem H.7 (Exact Binary Multiplication)**.** *Both extracted controllers produce the expert PTO on every valid binary-multiplication input. Both executions terminate with the $(m+n)$-bit representation of AB, including leading-zero padding, on row $n-1$ in columns $-n, \ldots, m-1$. The final head stops at $(-n-1, n-1)$.*

H.5.1 Shared Expert Behavior

Copying and Partial Products Write $b = b_{n-1} \cdots b_0$, so $B = \sum_{i=0}^{n-1} b_i 2^i$. Assign global weight 2^{m-1-z} to column z. Column $m-1$ carries the units digit of the final product.

Lemma H.8 (Copy Staircase)**.** *Before product formation, the expert constructs exactly n occupied rows. For $0 \leq i < n$, row i contains*

$$a\,x\,b_{n-1} \cdots b_i \quad \textit{starting in column } -i. \tag{15}$$

The last row contains one right-operand bit. All remaining cells contain blanks.

Proof. Row 0 contains the input. On row i, the expert scans to the first blank, steps left to b_i, then steps left again. A digit at this position implies at least two remaining right-operand bits. The expert copies every symbol except b_i one row down and one column left, proceeding from right to left. Every destination contains a blank before writing; the source word remains unchanged.

The first descent reaches a blank on the new row. Each later descent reaches the copied right neighbor: a digit, except directly below the last digit of a, where the copied separator x occupies the lower cell. The cell below the source separator contains a digit. After copying the first digit of a, the expert reads the blank immediately to the left and descends to the first digit of the new row.

The new row satisfies (15) for $i+1$, with one fewer right-operand bit. When one right-operand bit remains, the second left step reads x and starts product formation. Thus $n-1$ copies produce exactly n occupied rows. □

Lemma H.9 (Aligned Partial Products)**.** *After product formation, row i contains the $(m+1)$-bit representation of Ab_i in columns $-i-1,\ldots,m-i-1$, with a leading 0. In global column weights, row i represents Ab_i2^i. All remaining cells contain blanks.*

Proof. The expert forms products from row $n-1$ upwards. On row i, the expert erases the rightmost factor bit b_i, the remaining factor prefix, and x, then walks left across a. For $b_i=0$, the expert writes zeros; for $b_i=1$, the expert preserves the digits of a. Both branches write one additional 0 on the left blank. The resulting word has the claimed support and value, with a leading zero because $Ab_i<2^m$.

The expert scans right to the blank at column $m-i$ and moves up. For $i>0$, the upper cell contains the last digit of a on row $i-1$; the next cell contains x. The expert scans the remaining factor prefix, selects the last bit, and repeats. Each upper prefix contains at least two bits, so another digit follows the selected bit during leftward erasure. For $i=0$, the upper cell contains a blank, ending product formation and starting summation on row 0. □

Both passes terminate: each copy shortens the remaining factor prefix, and product formation visits each of the n rows once. Every horizontal scan crosses a finite word.

A Fixed-Width Addition Invariant

Lemma H.10 (Accumulated Product)**.** *After merging through row k, $0\le k<n$, all rows above k contain only blanks. Row k contains a contiguous $(m+k+1)$-bit accumulator on columns $-k-1,\ldots,m-1$, with value*

$$S_k = A\sum_{i=0}^{k} b_i 2^i.$$

Rows below k retain the partial products from Lemma H.9. At the inspection of the next row, the expert occupies state 20 with head $(-k-2,k+1)$.

Proof. For $k=0$, the initial left scan crosses the product on row 0, reads the leading 0, reaches the blank at column -2, and descends.

If row $k+1$ exists, the inspected cell contains the leading 0 at column $-k-2$. The expert scans right across the lower product to the first blank, at column $m-k-1$. Moving up reaches an accumulator digit because

$$-k-1 \;\le\; m-k-1 \;\le\; m-1.$$

The expert scans the remaining upper digits, reaches the right blank, and steps left to column $m-1$. The lower cell contains a blank because the original lower product ends at $m-k-2$. The first column therefore adds the upper bit to zero without a carry.

The expert processes the remaining columns from right to left. For upper bit u, lower bit v (with a blank counting as zero), and incoming carry c, the update satisfies

$$u+v+c = w+2c', \qquad w\in\{0,1\},\quad c'\in\{0,1\}. \tag{16}$$

The expert erases the upper bit, writes w below, and stores c' in the control state. The processed lower suffix contains the correct sum suffix, the unprocessed cells retain the previous values, and the carry records the contribution to the next column. Equation (16) preserves this invariant.

The original lower word extends one column farther left than the upper word, with a leading 0 at $-k-2$. After consuming the upper word, carry zero preserves this bit, whereas carry one changes the bit to 1 and clears. The expert reaches the blank at $-k-3$ on row $k+1$ and descends. Row k now contains only blanks, and row $k+1$ represents $S_k + Ab_{k+1}2^{k+1} = S_{k+1}$ on the claimed support. □

Lemma H.11 (Carry Confinement)**.** *During an upper-digit update with incoming carry* 1*, the lower cell contains an original binary digit. If the carry outlives the upper word, the next lower cell contains the original leading* 0*.*

Proof. A blank gap lies to the right of the original lower product. The first column has carry zero, and $u + 0 + 0 \leq 1$, so no carry originates in the gap. A carry first arises inside the lower digit block. Further leftward steps remain inside that block until the leading zero. The upper word ends one column before that zero, where $0 + 0 + 1 = 1$ clears the carry. □

Each merge consumes finitely many upper digits and at most one extra carry column. After $n-1$ merges, state 20 reads the blank below the final row and moves up to state 30, which emits the final S action and enters halt state 31. Lemma H.10 gives the $(m+n)$-bit product and final head $(-n-1, n-1)$. All other cells contain blanks.

H.5.2 Metric-Extraction Certificate

The observation protocol refines expert states by geometric context. Each phase records the permitted read symbols and successor phases. Section H.5.2 gives the complete protocol together with the controller-state relation; the shared tables avoid repeating the same phase definitions.

Lemma H.12 (Protocol Coverage)**.** *Every expert execution on a well-formed multiplication follows the* 67 *phases in Section H.5.2, from phase* 0 *to phase* 66*.*

Proof. Lemma H.8 supplies the copy-phase conditions; Lemma H.9 supplies the product-phase conditions. Lemma H.10 and Lemma H.11 supply the alignment, addition, carry and boundary conditions. The explanations below the three tables account for the first and last cell readings in each phase. Induction over the individual transitions, the copied rows, and the merged rows establishes coverage. The finite-pass arguments in the expert invariants give termination in the final phase. □

The Copying Phases The following three tables define a finite protocol and a simulation relation. Phase i pairs expert state g_i with controller state $q \in Q_i$ on an identical grid and head. Each set Q_i contains the single listed state. An entry

$$x : (j, w, M)$$

means: read x, write w, move M, and advance to phase j. Here $d \in \{0, 1\}$ denotes either digit, consistently within each entry. The index j denotes a phase. Successors comprise expert state g_j and controller state $q' \in Q_j$. The listed transitions specify the required local identities. The protocol starts at phase 0.

i	Phase	g_i	Q_i	Read : (next phase, write, move)
0	First left-operand digit	0	$\{0\}$	$d : (1, d, \mathsf{R})$
1	Scan left operand	0	$\{1\}$	$d : (1, d, \mathsf{R})$ $x : (2, x, \mathsf{R})$
2	First right-operand digit	0	$\{2\}$	$d : (3, d, \mathsf{R})$
3	Scan right operand	0	$\{3\}$	$d : (3, d, \mathsf{R})$ $\lambda : (4, \lambda, \mathsf{L})$
4	Skip last right-operand digit	1	$\{4\}$	$d : (5, d, \mathsf{L})$
5	Copy more, or start products	2	$\{5\}$	$0 : (6, 0, \mathsf{D})$ $1 : (8, 1, \mathsf{D})$ $x : (22, x, \mathsf{R})$
6	First copied 0: descend	3	$\{52\}$	$\lambda : (7, \lambda, \mathsf{L})$
7	First copied 0: write	6	$\{53\}$	$\lambda : (10, 0, \mathsf{U})$
8	First copied 1: descend	4	$\{6\}$	$\lambda : (9, \lambda, \mathsf{L})$
9	First copied 1: write	7	$\{7\}$	$\lambda : (10, 1, \mathsf{U})$
10	Continue after first copy	9	$\{8\}$	$0 : (11, 0, \mathsf{D})$ $1 : (13, 1, \mathsf{D})$ $x : (16, x, \mathsf{D})$
11	Copy 0 above a lower digit	3	$\{16\}$	$d : (12, d, \mathsf{L})$

i	Phase	g_i	Q_i	Read : (next phase, write, move)			
12	Write copied 0	6	$\{17\}$	$\lambda : (15, 0, \mathsf{U})$			
13	Copy 1 above a lower digit	4	$\{9\}$	$d : (14, d, \mathsf{L})$			
14	Write copied 1	7	$\{10\}$	$\lambda : (15, 1, \mathsf{U})$			
15	Continue the copy	9	$\{11\}$	$0 : (11, 0, \mathsf{D})$	$1 : (13, 1, \mathsf{D})$	$x : (16, x, \mathsf{D})$	$\lambda : (21, \lambda, \mathsf{D})$
16	Descend below operator	5	$\{12\}$	$d : (17, d, \mathsf{L})$			
17	Write shifted operator	8	$\{13\}$	$\lambda : (18, x, \mathsf{U})$			
18	Copy last left-operand digit	9	$\{14\}$	$0 : (19, 0, \mathsf{D})$	$1 : (20, 1, \mathsf{D})$		
19	Copied 0 above lower operator	3	$\{55\}$	$x : (12, x, \mathsf{L})$			
20	Copied 1 above lower operator	4	$\{15\}$	$x : (14, x, \mathsf{L})$			
21	Begin scan on copied row	0	$\{0\}$	$d : (1, d, \mathsf{R})$			

Why the Listed Read Symbols Cover Every Copy. The nonempty operands ensure that the initial scan reads a digit, the separator, and at least one factor digit before reaching a blank. The second left step sees either a digit to copy or x when only one factor bit remains. Lemma H.8 identifies the three possible cells below a copied digit: blank for the first copy, x immediately after copying the separator, and a digit otherwise. Every destination contains a blank. Copying the first digit of a exposes the left blank; restarting on the new row therefore reads a digit. These facts establish the table's geometric restrictions.

The protocol permits arbitrary operand lengths and may admit additional observation sequences. Every valid execution follows permitted branches; checking all such branches therefore establishes the required local agreement.

The One-Bit Product Phases Phase i pairs expert state g_i with controller state $q \in Q_i$. An edge $x : (j, w, M)$ reads x, writes w, performs motion M, and advances to phase j. The product phases preserve the exact leading-zero convention of the expert.

i	Phase	g_i	Q_i	Read : (next phase, write, move)	
22	Read the last remaining factor bit	10	$\{18\}$	$0 : (23, \lambda, \mathsf{L})$	$1 : (26, \lambda, \mathsf{L})$
23	Erase operator for factor 0	11	$\{56\}$	$x : (24, \lambda, \mathsf{L})$	
24	First digit of product by 0	13	$\{32\}$	$d : (25, 0, \mathsf{L})$	
25	Continue product by 0	13	$\{33\}$	$d : (25, 0, \mathsf{L})$	$\lambda : (29, 0, \mathsf{R})$
26	Erase operator for factor 1	12	$\{19\}$	$x : (27, \lambda, \mathsf{L})$	
27	First digit of product by 1	14	$\{20\}$	$d : (28, d, \mathsf{L})$	
28	Continue product by 1	14	$\{21\}$	$d : (28, d, \mathsf{L})$	$\lambda : (29, 0, \mathsf{R})$
29	Scan completed partial product	15	$\{22\}$	$d : (29, d, \mathsf{R})$	$\lambda : (30, \lambda, \mathsf{U})$
30	Inspect the row above	16	$\{23\}$	$d : (31, d, \mathsf{R})$	$\lambda : (39, \lambda, \mathsf{D})$
31	Cross upper operator	17	$\{24\}$	$x : (32, x, \mathsf{R})$	
32	First bit in upper factor prefix	17	$\{25\}$	$d : (33, d, \mathsf{R})$	
33	Scan upper factor prefix	17	$\{26\}$	$d : (33, d, \mathsf{R})$	$\lambda : (34, \lambda, \mathsf{L})$
34	Read upper row factor bit	10	$\{27\}$	$0 : (35, \lambda, \mathsf{L})$	$1 : (37, \lambda, \mathsf{L})$
35	First remaining bit to erase (0)	11	$\{30\}$	$d : (36, \lambda, \mathsf{L})$	
36	Erase factor prefix (0)	11	$\{31\}$	$d : (36, \lambda, \mathsf{L})$	$x : (24, \lambda, \mathsf{L})$
37	First remaining bit to erase (1)	12	$\{28\}$	$d : (38, \lambda, \mathsf{L})$	
38	Erase factor prefix (1)	12	$\{29\}$	$d : (38, \lambda, \mathsf{L})$	$x : (27, \lambda, \mathsf{L})$

First and Subsequent Product Rows. The bottom row contains one factor bit. Erasing that bit moves the head directly onto x. Every higher row has at least two right-operand bits: after erasing the selected last bit, the expert erases another digit before reaching the separator. These facts justify the separate "single" and "first remaining bit" phases in the table.

Upper-Row Alignment. A finished product occupies $[-i - 1, m - i - 1]$ on row i. The right scan ends at column $m - i$. An existing upper row has the last left-operand digit at that column, followed immediately by x. After that separator comes a nonempty remaining right operand. If no row above exists, the expert reads a blank and starts summation. Lemma H.9 proves these conditions for every operand length, including operands containing only zeros.

Product-Scan Coverage. Multiplication by 0 turns each left-operand digit into 0; multiplication by 1 preserves each digit. Both passes have at least one digit to process and finish on a blank by writing a padding 0. The subsequent right scan sees only digits followed by a blank. A completed product row contains no separators.

Summation, Carry and the Final Stop These phases refine the expert's addition states by the position in a merge and the carry condition. This refinement supplies every required transition in the partial controller table on all well-formed multiplications.

i	Phase	g_i	Q_i	Read : (next phase, write, move)		
39	Start summing the product rows	18	$\{34\}$	$\lambda : (40, \lambda, \mathsf{L})$		
40	First digit of initial left scan	19	$\{35\}$	$0 : (41, 0, \mathsf{L})$	$1 : (42, 1, \mathsf{L})$	
41	Left scan after reading zero	19	$\{36\}$	$0 : (41, 0, \mathsf{L})$	$1 : (42, 1, \mathsf{L})$	$\lambda : (43, \lambda, \mathsf{D})$
42	Left scan after reading one	19	$\{35\}$	$0 : (41, 0, \mathsf{L})$	$1 : (42, 1, \mathsf{L})$	
43	Inspect fresh lower leading cell	20	$\{37\}$	$0 : (44, 0, \mathsf{R})$	$\lambda : (65, \lambda, \mathsf{U})$	
44	First digit after lower leading zero	21	$\{22\}$	$d : (45, d, \mathsf{R})$		
45	Scan the lower partial product	21	$\{22\}$	$d : (45, d, \mathsf{R})$	$\lambda : (46, \lambda, \mathsf{U})$	
46	First upper alignment digit	28	$\{23\}$	$d : (47, d, \mathsf{R})$		
47	Next upper alignment cell	28	$\{24\}$	$d : (48, d, \mathsf{R})$	$\lambda : (49, \lambda, \mathsf{L})$	
48	Finish aligning the upper row	28	$\{45\}$	$d : (48, d, \mathsf{R})$	$\lambda : (49, \lambda, \mathsf{L})$	
49	Read upper units digit	22	$\{38\}$	$0 : (50, \lambda, \mathsf{D})$	$1 : (51, \lambda, \mathsf{D})$	
50	Write units digit 0 below	24	$\{39\}$	$\lambda : (52, 0, \mathsf{U})$		
51	Write units digit 1 below	25	$\{54\}$	$\lambda : (52, 1, \mathsf{U})$		
52	Return above, carry zero	23	$\{40\}$	$\lambda : (53, \lambda, \mathsf{L})$		
53	Read upper bit, carry zero	22	$\{41\}$	$0 : (54, \lambda, \mathsf{D})$	$1 : (55, \lambda, \mathsf{D})$	$\lambda : (60, \lambda, \mathsf{D})$
54	Add value zero below	24	$\{42\}$	$d : (52, d, \mathsf{U})$	$\lambda : (52, 0, \mathsf{U})$	
55	Add value one below	25	$\{46\}$	$\lambda : (52, 1, \mathsf{U})$	$0 : (52, 1, \mathsf{U})$	$1 : (56, 0, \mathsf{U})$
56	Return above, carry one	27	$\{47\}$	$\lambda : (57, \lambda, \mathsf{L})$		
57	Read upper bit, carry one	26	$\{48\}$	$0 : (58, \lambda, \mathsf{D})$	$1 : (59, \lambda, \mathsf{D})$	$\lambda : (62, \lambda, \mathsf{D})$
58	Add one inside lower digit block	25	$\{50\}$	$0 : (52, 1, \mathsf{U})$	$1 : (56, 0, \mathsf{U})$	
59	Add two inside lower digit block	29	$\{49\}$	$d : (56, d, \mathsf{U})$		
60	Preserve unneeded leading zero	19	$\{43\}$	$0 : (61, 0, \mathsf{L})$		
61	Blank to the left of merged row	19	$\{44\}$	$\lambda : (43, \lambda, \mathsf{D})$		
62	Apply final carry to leading zero	25	$\{57\}$	$0 : (63, 1, \mathsf{U})$		
63	Return after final carry	23	$\{58\}$	$\lambda : (64, \lambda, \mathsf{L})$		
64	Step down left of merged row	22	$\{59\}$	$\lambda : (61, \lambda, \mathsf{D})$		
65	Emit final stop	30	$\{51\}$	$\lambda : (66, \lambda, \mathsf{S})$		
66	Halted	31	$\{60\}$	–		

Boundary Restrictions. The initial left scan can encounter a blank only after reading the product's leading 0. The next inspection reads either a blank or the fresh row's leading 0. Alignment always starts on an upper digit, and the first lower units cell contains a blank. Lemma H.11 excludes lower blanks in the carry-one digit phases and forces 0 for the final carry extension. With no final carry, the expert preserves the untouched leading 0; after either branch, the next cell to the left of the result contains a blank. The final stop also reads a blank.

These facts, together with contiguity of every accumulator, justify all read restrictions in this table. The restrictions exclude every missing transition that could otherwise obstruct a valid multiplication.

Local Transition Identity. For each of the 130 listed read/action cases and the unique $q \in Q_i$, direct comparison with the two transition maps gives

$$\delta_{\mathcal{E}}(g_i, x) = (g_j, w, M), \qquad \delta_{\mathcal{C}}(q, x) = (q', w, M), \qquad q' \in Q_j. \tag{17}$$

Every required entry exists. A transition reaches a halt state exactly when $j = 66$, with motion S. The certificate covers all 119 defined nonterminal controller entries. The remaining four entries contain halt loops.

H.5.3 Annealing Certificate

The annealed controller reuses internal states more extensively. The refined protocol remembers unchanged factor bits and distinguishes one-digit scans from longer scans; these annotations belong to the proof, without adding memory to either controller. The observation protocol refines expert states by geometric context. Each phase records the permitted read symbols and successor phases. Section H.5.3 gives the complete protocol together with the controller-state relation; the shared tables avoid repeating the same phase definitions.

Lemma H.13 (Protocol Coverage)**.** *Every expert execution on a well-formed multiplication follows the* 94 *phases in Section H.5.3, from phase* 0 *to phase* 93*.*

Proof. The copy and product invariants (Lemma H.8–Lemma H.9) give all copying and product-reading conditions. The scan annotations remember unchanged cells and distinguish one digit from at least two; the explanations in Section H.5.3–Section H.5.3 prove these refinements. Lemma H.10–Lemma H.11 and the blank-gap argument in Section H.5.3 supply the addition and boundary conditions. Induction over each transition, each copied row and each merged row establishes coverage. The finite-pass arguments in the expert invariants give termination in phase 93. □

Copying and Remembered Factor Bits The next four tables define a finite protocol and simulation relation. Phase i pairs expert state g_i with a controller state $q \in Q_i$, with identical grids and heads. An entry $x : (j, w, M)$ means read x, write w, move M, and advance to *phase* j. The symbol $d \in \{0, 1\}$ means either digit, consistently within that entry. For *each* $q \in Q_i$, the actual successor lies in Q_j.

i	Phase	g_i	Q_i	Read : (next phase, write, move)
0	First left-operand digit	0	$\{0\}$	$d : (1, d, \mathsf{R})$
1	Scan left operand	0	$\{0, 24\}$	$d : (1, d, \mathsf{R})$ $x : (2, x, \mathsf{R})$
2	First right-operand digit	0	$\{7\}$	$0 : (3, 0, \mathsf{R})$ $1 : (4, 1, \mathsf{R})$
3	Scan factor; last bit 0	0	$\{48\}$	$0 : (3, 0, \mathsf{R})$ $1 : (4, 1, \mathsf{R})$ $\lambda : (5, \lambda, \mathsf{L})$
4	Scan factor; last bit 1	0	$\{48\}$	$0 : (3, 0, \mathsf{R})$ $1 : (4, 1, \mathsf{R})$ $\lambda : (6, \lambda, \mathsf{L})$
5	Revisit last factor bit 0	1	$\{14\}$	$0 : (7, 0, \mathsf{L})$
6	Revisit last factor bit 1	1	$\{14\}$	$1 : (8, 1, \mathsf{L})$
7	Copy or use remembered bit 0	2	$\{49\}$	$0 : (9, 0, \mathsf{D})$ $1 : (11, 1, \mathsf{D})$ $x : (25, x, \mathsf{R})$
8	Copy or use remembered bit 1	2	$\{19\}$	$0 : (9, 0, \mathsf{D})$ $1 : (11, 1, \mathsf{D})$ $x : (26, x, \mathsf{R})$
9	First copied 0: descend	3	$\{16\}$	$\lambda : (10, \lambda, \mathsf{L})$
10	First copied 0: write	6	$\{13\}$	$\lambda : (13, 0, \mathsf{U})$
11	First copied 1: descend	4	$\{4\}$	$\lambda : (12, \lambda, \mathsf{L})$
12	First copied 1: write	7	$\{41\}$	$\lambda : (13, 1, \mathsf{U})$
13	Continue after first copy	9	$\{35\}$	$0 : (14, 0, \mathsf{D})$ $1 : (16, 1, \mathsf{D})$ $x : (19, x, \mathsf{D})$
14	Copy 0 above a lower digit	3	$\{44, 47\}$	$d : (15, d, \mathsf{L})$
15	Write copied 0	6	$\{9, 49\}$	$\lambda : (18, 0, \mathsf{U})$
16	Copy 1 above a lower digit	4	$\{26\}$	$d : (17, d, \mathsf{L})$
17	Write copied 1	7	$\{2, 3\}$	$\lambda : (18, 1, \mathsf{U})$
18	Continue the copy	9	$\{34\}$	$0 : (14, 0, \mathsf{D})$ $1 : (16, 1, \mathsf{D})$ $x : (19, x, \mathsf{D})$ $\lambda : (24, \lambda, \mathsf{D})$
19	Descend below operator	5	$\{45\}$	$d : (20, d, \mathsf{L})$
20	Write shifted operator	8	$\{5\}$	$\lambda : (21, x, \mathsf{U})$
21	Copy last left-operand digit	9	$\{34\}$	$0 : (22, 0, \mathsf{D})$ $1 : (23, 1, \mathsf{D})$
22	Copied 0 above lower operator	3	$\{44\}$	$x : (15, x, \mathsf{L})$
23	Copied 1 above lower operator	4	$\{26\}$	$x : (17, x, \mathsf{L})$
24	Begin scan on copied row	0	$\{36\}$	$d : (1, d, \mathsf{R})$

Coverage. Both operands contain at least one digit. The initial scan therefore meets a digit, x, and a right-operand digit before a blank. Phases 3–8 remember the last factor bit: the scan's last digit, the digit revisited by the first left step, and (if only one factor bit remains) the digit selected for multiplication all refer to the same unchanged cell.

Lemma H.8 proves the remaining restrictions: the first descent sees a blank; subsequent descents see the already copied right-neighbor digit, except below the last left-operand digit, where the lower cell

contains x. Every destination contains a blank. Copying x exposes a source digit. Restarting on a copied row also reads a digit. These conditions hold for arbitrary operand lengths and values.

Forming and Scanning Partial Products The product scan starts just to the right of the new padding zero. "One" denotes a scan after exactly one product digit; the remaining scan phases follow at least two product digits. Each scan also records the most recently read digit.

i	Phase	g_i	Q_i	Read : (next phase, write, move)		
25	Select remembered factor bit 0	10	$\{10\}$	$0 : (27, \lambda, \mathsf{L})$		
26	Select remembered factor bit 1	10	$\{3\}$	$1 : (28, \lambda, \mathsf{L})$		
27	Erase operator for factor 0	11	$\{10\}$	$x : (29, \lambda, \mathsf{L})$		
28	Erase operator for factor 1	12	$\{4\}$	$x : (31, \lambda, \mathsf{L})$		
29	First digit of product by 0	13	$\{25\}$	$d : (30, 0, \mathsf{L})$		
30	Continue product by 0	13	$\{25\}$	$d : (30, 0, \mathsf{L})$	$\lambda : (33, 0, \mathsf{R})$	
31	First digit of product by 1	14	$\{6\}$	$d : (32, d, \mathsf{L})$		
32	Continue product by 1	14	$\{6\}$	$d : (32, d, \mathsf{L})$	$\lambda : (33, 0, \mathsf{R})$	
33	First digit after product padding	15	$\{15, 36\}$	$0 : (34, 0, \mathsf{R})$	$1 : (35, 1, \mathsf{R})$	
34	After one product digit 0	15	$\{0\}$	$0 : (36, 0, \mathsf{R})$	$1 : (37, 1, \mathsf{R})$	$\lambda : (38, \lambda, \mathsf{U})$
35	After one product digit 1	15	$\{24\}$	$0 : (36, 0, \mathsf{R})$	$1 : (37, 1, \mathsf{R})$	$\lambda : (39, \lambda, \mathsf{U})$
36	Product scan; last digit 0	15	$\{24\}$	$0 : (36, 0, \mathsf{R})$	$1 : (37, 1, \mathsf{R})$	$\lambda : (40, \lambda, \mathsf{U})$
37	Product scan; last digit 1	15	$\{24\}$	$0 : (36, 0, \mathsf{R})$	$1 : (37, 1, \mathsf{R})$	$\lambda : (41, \lambda, \mathsf{U})$
38	Inspect above one-digit product 0	16	$\{39\}$	$d : (42, d, \mathsf{R})$	$\lambda : (52, \lambda, \mathsf{D})$	
39	Inspect above one-digit product 1	16	$\{17\}$	$d : (42, d, \mathsf{R})$	$\lambda : (53, \lambda, \mathsf{D})$	
40	Inspect above; last digit 0	16	$\{17\}$	$d : (42, d, \mathsf{R})$	$\lambda : (54, \lambda, \mathsf{D})$	
41	Inspect above; last digit 1	16	$\{17\}$	$d : (42, d, \mathsf{R})$	$\lambda : (55, \lambda, \mathsf{D})$	
42	Cross upper operator	17	$\{8, 43\}$	$x : (43, x, \mathsf{R})$		
43	First bit in upper factor prefix	17	$\{16\}$	$0 : (44, 0, \mathsf{R})$	$1 : (45, 1, \mathsf{R})$	
44	Scan factor; last bit 0	17	$\{16\}$	$0 : (44, 0, \mathsf{R})$	$1 : (45, 1, \mathsf{R})$	$\lambda : (46, \lambda, \mathsf{L})$
45	Scan factor; last bit 1	17	$\{16\}$	$0 : (44, 0, \mathsf{R})$	$1 : (45, 1, \mathsf{R})$	$\lambda : (47, \lambda, \mathsf{L})$
46	Select remembered factor bit 0	10	$\{13\}$	$0 : (48, \lambda, \mathsf{L})$		
47	Select remembered factor bit 1	10	$\{13\}$	$1 : (50, \lambda, \mathsf{L})$		
48	First remaining bit to erase (0)	11	$\{27\}$	$d : (49, \lambda, \mathsf{L})$		
49	Erase factor prefix (0)	11	$\{10\}$	$d : (49, \lambda, \mathsf{L})$	$x : (29, \lambda, \mathsf{L})$	
50	First remaining bit to erase (1)	12	$\{22\}$	$d : (51, \lambda, \mathsf{L})$		
51	Erase factor prefix (1)	12	$\{1\}$	$d : (51, \lambda, \mathsf{L})$	$x : (31, \lambda, \mathsf{L})$	

Coverage. Lemma H.9 gives a nonempty block of m product digits after the padding zero. This justifies the first-digit phase and the split into exactly one or at least two scanned digits. At the right blank, the most recently read digit equals the units digit.

The bottom row contains one factor bit; erasing that bit exposes x. Every higher row contains at least two factor bits, so another digit precedes x during erasure. An upward return from a completed row reaches either a blank or the last left-operand digit followed by x. The upper factor scan records the last bit, which remains unchanged until selection. All these facts follow from the copy staircase.

Starting the Sum and Aligning a Row These phases retain the last product digit until the next visit and distinguish a one-digit left operand from a longer operand. The remembered bit and length class serve as proof annotations without adding machine registers.

i	Phase	g_i	Q_i	Read : (next phase, write, move)
52	Begin one-digit sum; last bit 0	18	$\{20\}$	$\lambda : (56, \lambda, \mathsf{L})$
53	Begin one-digit sum; last bit 1	18	$\{42\}$	$\lambda : (57, \lambda, \mathsf{L})$
54	Begin sum; remember last digit 0	18	$\{42\}$	$\lambda : (58, \lambda, \mathsf{L})$
55	Begin sum; remember last digit 1	18	$\{42\}$	$\lambda : (59, \lambda, \mathsf{L})$
56	Revisit sole product digit 0	19	$\{28\}$	$0 : (60, 0, \mathsf{L})$
57	Revisit sole product digit 1	19	$\{11\}$	$1 : (60, 1, \mathsf{L})$
58	Revisit product units digit 0	19	$\{11\}$	$0 : (62, 0, \mathsf{L})$
59	Revisit product units digit 1	19	$\{11\}$	$1 : (62, 1, \mathsf{L})$

i	Phase	g_i	Q_i	Read : (next phase, write, move)		
60	Read the sole leading zero	19	$\{2\}$	$0 : (61, 0, \mathsf{L})$		
61	Left boundary after sole digit	19	$\{23\}$	$\lambda : (66, \lambda, \mathsf{D})$		
62	Second digit of initial left scan	19	$\{2, 26\}$	$d : (63, d, \mathsf{L})$		
63	Third digit of initial left scan	19	$\{2, 23\}$	$0 : (64, 0, \mathsf{L})$	$1 : (65, 1, \mathsf{L})$	
64	Left scan after reading zero	19	$\{23\}$	$0 : (64, 0, \mathsf{L})$	$1 : (65, 1, \mathsf{L})$	$\lambda : (66, \lambda, \mathsf{D})$
65	Left scan after reading one	19	$\{23\}$	$0 : (64, 0, \mathsf{L})$	$1 : (65, 1, \mathsf{L})$	
66	Inspect fresh lower leading cell	20	$\{12, 36\}$	$0 : (67, 0, \mathsf{R})$	$\lambda : (92, \lambda, \mathsf{U})$	
67	First digit after lower leading zero	21	$\{0\}$	$d : (68, d, \mathsf{R})$		
68	Scan the lower partial product	21	$\{24\}$	$d : (68, d, \mathsf{R})$	$\lambda : (69, \lambda, \mathsf{U})$	
69	First upper alignment digit	28	$\{17\}$	$d : (70, d, \mathsf{R})$		
70	Next upper alignment cell	28	$\{8\}$	$d : (71, d, \mathsf{R})$	$\lambda : (72, \lambda, \mathsf{L})$	
71	Finish aligning the upper row	28	$\{4, 8\}$	$d : (71, d, \mathsf{R})$	$\lambda : (72, \lambda, \mathsf{L})$	

Initial Left-Scan Coverage. On the top row, the product's units digit has not changed between the right scan and the first left step of summation. If $m = 1$, the full padded word equals $0d$: scanning left reads d, then 0, then a blank (phases 56–57, 60–61). If $m \geq 2$, the padded word has at least three digits, so two more digits must follow the first one (phases 62–63). In both cases the padding zero immediately precedes the left blank.

Alignment Coverage. Lemma H.10 gives either a blank in the inspected lower cell (no remaining row) or the next row's leading 0. After that zero comes at least one digit. The upward move from the lower right boundary reaches an upper digit, so alignment begins inside the upper word. The expert scans to the right blank and then returns left to the units digit. The first lower units cell contains a blank. These facts justify every alignment branch here and the first three phases in the next certificate subsection.

Column Addition, Carries and Stopping Separate phases track carry-zero returns from lower blanks and lower digits. This distinction prevents an impossible early end of the upper word from entering the proof protocol.

i	Phase	g_i	Q_i	Read : (next phase, write, move)		
72	Read upper units digit	22	$\{32, 41\}$	$0 : (73, \lambda, \mathsf{D})$	$1 : (74, \lambda, \mathsf{D})$	
73	Write units digit 0 below	24	$\{27, 46\}$	$\lambda : (88, 0, \mathsf{U})$		
74	Write units digit 1 below	25	$\{22\}$	$\lambda : (88, 1, \mathsf{U})$		
75	Return above, carry zero	23	$\{50\}$	$\lambda : (76, \lambda, \mathsf{L})$		
76	Read upper bit, carry zero	22	$\{37\}$	$0 : (77, \lambda, \mathsf{D})$	$1 : (78, \lambda, \mathsf{D})$	$\lambda : (83, \lambda, \mathsf{D})$
77	Add value zero below	24	$\{30\}$	$d : (75, d, \mathsf{U})$	$\lambda : (90, 0, \mathsf{U})$	
78	Add value one below	25	$\{29\}$	$\lambda : (90, 1, \mathsf{U})$	$0 : (75, 1, \mathsf{U})$	$1 : (79, 0, \mathsf{U})$
79	Return above, carry one	27	$\{10\}$	$\lambda : (80, \lambda, \mathsf{L})$		
80	Read upper bit, carry one	26	$\{38\}$	$0 : (81, \lambda, \mathsf{D})$	$1 : (82, \lambda, \mathsf{D})$	$\lambda : (85, \lambda, \mathsf{D})$
81	Add one inside lower digit block	25	$\{29\}$	$0 : (75, 1, \mathsf{U})$	$1 : (79, 0, \mathsf{U})$	
82	Add two inside lower digit block	29	$\{9\}$	$d : (79, d, \mathsf{U})$		
83	Preserve unneeded leading zero	19	$\{20\}$	$0 : (84, 0, \mathsf{L})$		
84	Blank to the left of merged row	19	$\{31, 34\}$	$\lambda : (66, \lambda, \mathsf{D})$		
85	Apply final carry to leading zero	25	$\{42\}$	$0 : (86, 1, \mathsf{U})$		
86	Return after final carry	23	$\{47\}$	$\lambda : (87, \lambda, \mathsf{L})$		
87	Step down left of merged row	22	$\{33\}$	$\lambda : (84, \lambda, \mathsf{D})$		
88	Return after units digit	23	$\{18, 21\}$	$\lambda : (89, \lambda, \mathsf{L})$		
89	Read the second upper digit	22	$\{21\}$	$0 : (77, \lambda, \mathsf{D})$	$1 : (78, \lambda, \mathsf{D})$	
90	Return from the lower blank gap	23	$\{21\}$	$\lambda : (91, \lambda, \mathsf{L})$		
91	Upper digit after lower blank gap	22	$\{21\}$	$0 : (77, \lambda, \mathsf{D})$	$1 : (78, \lambda, \mathsf{D})$	
92	Emit final stop	30	$\{35, 40\}$	$\lambda : (93, \lambda, \mathsf{S})$		
93	Halted	31	$\{51\}$	–		

The Lower Blank Gap. While merging row k into row $k + 1$, the upper word begins at $-k - 1$; the original lower word ends at $m - k - 2 \geq -k - 1$. A blank below an upper digit must therefore lie

to the right of that lower endpoint. The next column left still has index at least $-k-1$, so the next upper reading contains a digit. This proves the restriction in phase 91. Independently, the upper word has length $m+k+1 \geq 2$, proving the second-digit restriction in phase 89.

Carry and Boundary Coverage. Lemma H.11 excludes lower blanks in the carry-one digit phases and forces 0 for the final carry extension. Returning above a written lower digit reads the upper blank just erased. When the upper word ends, the next lower cell contains the original leading zero: carry zero preserves this bit, whereas carry one changes the bit to 1. Both branches reach the blank to the left of the merged row. When no next row remains, the upward move reaches a blank, and the expert emits the final S. Phase 93 has no further PTO actions.

Local Transition Identity. For every listed input branch $x : (j, w, M)$ and every $q \in Q_i$, direct comparison with the two transition maps gives

$$\delta_{\mathcal{E}}(g_i, x) = (g_j, w, M), \qquad \delta_{\mathcal{C}}(q, x) = (q', w, M) \quad \text{for some } q' \in Q_j. \tag{18}$$

The certificate contains 109 phase–state pairs and 201 local checks, covering 133 distinct nonterminal controller entries. A transition reaches a halt state exactly when $j = 93$, with $M = S$. Initially $g_0 = 0$ and $Q_0 = \{0\}$. Protocol coverage excludes the remaining controller entries on valid inputs.

Proof of Theorem H.7. The copy, partial-product, and accumulation invariants establish the claimed product and termination for the expert. The metric protocol covers every valid execution through 67 phases; the refined annealing protocol covers every valid execution through 94 phases. The respective transition maps satisfy all 130 and 201 local identities, including the initial and terminal conditions. Applying Lemma H.2 separately to the two protocols proves both equivalence claims. □

H.6 Decimal Addition

Let $D = \{0, \ldots, 9\}$ and consider $a + b$, with $a \in D^m$, $b \in D^n$ and $m, n \geq 1$. Write $A = \mathrm{val}_{10}(a)$ and $B = \mathrm{val}_{10}(b)$. The expert and metric-extracted controller start in state 0 and halt in states 18 and 51, respectively.

Theorem H.14 (Exact Decimal Addition)**.** *The metric-extracted controller produces the expert PTO on every valid decimal-addition input, with identical grids and head positions at every step.*

Accumulator and Marker Invariant Write $B = \sum_{j=0}^{n-1} b_j 10^j$, with b_0 the rightmost digit. The expert scans to the separator at column m, steps left and down, writes a working $+$ below column $m-1$, and enters state 16 above the marker.

Lemma H.15 (Round Invariant)**.** *After processing k right-operand digits, $0 \leq k \leq n$, the expert enters state 16 at $(p_k, 0)$, where $p_k = m - 1 - k$. At this point:*

- *Row 1 contains exactly one nonblank symbol: a + at column p_k.*
- *Row 0 contains a contiguous accumulator on $[\ell_k, m-1]$, with value $C_k = A + \sum_{j<k} b_j 10^j$ and left edge $\ell_k \leq p_k + 1$. Column $m-1$ carries the units digit.*
- *The separator remains at column m. The unprocessed prefix of b occupies columns $m+1, \ldots, m+n-k$. All remaining cells contain blanks.*

Proof. Initialization gives $k = 0$, $\ell_0 = 0$, $C_0 = A$, and $p_0 = m - 1$. State 16 preserves the digit at p_k, or writes 0 on a blank. A blank at p_k forces $\ell_k = p_k + 1$, so zero extension preserves contiguity.

For $k < n$, the right scan reaches the blank after the unprocessed prefix, steps left to digit b_k at $r_k = m + n - k$, erases b_k, and descends. The expert transports b_k left along row 1 to the marker. The distance satisfies

$$r_k - p_k = n + 1 \geq 2. \tag{19}$$

Consequently, the first two transport steps read blanks. At the marker, the expert ascends and adds b_k to the accumulator digit e at column p_k. If $e + b_k < 10$, the expert writes $e + b_k$ and descends. Otherwise, the expert writes $e + b_k - 10$, moves left, and propagates one carry. Each digit 9 changes to 0; the first smaller digit increases by one, or a blank changes to 1. The update adds exactly $b_k 10^k$ and preserves contiguity.

The expert returns along row 1, erases the marker, steps left, writes the next marker at $p_{k+1} = p_k - 1$, and ascends into state 16. The earlier zero extension gives $\ell_{k+1} \le p_k = p_{k+1} + 1$. The accumulator now has value C_{k+1}, establishing the next round. □

Lemma H.16 (Termination and Final Configuration)**.** *The expert terminates with only the padded base-*10 *representation of $A + B$ on row* 0*, ending at column $m - 1$. The final head stops at* $(m - n - 1, 0)$.

Writing $\ell_{10}(s)$ for the decimal digit count, with $\ell_{10}(0) = 1$, the output width equals $L = \max\{m, n + 1, \ell_{10}(A + B)\}$.

Proof. Each round consumes one of the n right-operand digits. Every scan crosses a finite interval, and every carry reaches a digit below 9 or a blank. After round n, state 16 ensures a digit at p_n. The next right scan returns to the separator. State 3 erases the separator and descends. State 17 walks left, erases the final marker, ascends, preserves the accumulator digit, and stops. The invariant gives $C_n = A + B$, and cleanup removes all work symbols.

The original m columns persist, and marker extension reaches p_n, ensuring at least $n + 1$ digits. Every additional carry extension contributes a digit required by the current numerical value. Since partial sums never decrease, the final width equals L. □

The observation protocol refines expert states by geometric context. Each phase records the permitted read symbols and successor phases. Section H.6 gives the complete protocol together with the controller-state relation; the shared tables avoid repeating the same phase definitions.

Lemma H.17 (Protocol Coverage)**.** *Every expert execution on a valid decimal addition grid follows the* 59 *phases in Section H.6, starting at phase* 0 *and ending at phase* 58*.*

Proof. Initialization satisfies the first phases because both operands contain a digit. Lemma H.15 gives the contiguous accumulator, the ensured digit, the single working marker and the remaining right-operand prefix in every round. The explanations in Section H.6 and Section H.6 establish all first-cell, boundary, transport and return readings. The carry rule processes exactly the digits permitted in phase 50. Induction over the transitions of each round preserves these conditions. Lemma H.16 establishes finite cleanup and the final phase. □

Scanning, Carrying, and Cleanup A proof phase records geometric context. Phase i pairs expert state g_i with controller state $q \in Q_i$, with identical grids and heads. An entry $x : (j, w, M)$ means read x, write w, move M, and advance to phase j. The index j denotes a phase. Each Q_i lists the related controller states.

Here $e \in D$ and $h \in \{0, \dots, 8\}$. A digit denotes the corresponding symbol when read or written. Phases 10–49 comprise ten digit-transport families in the next certificate subsection, with $F_d = 10 + 4d$.

i	Phase	g_i	Q_i	Read : (next phase, write, move)
0	First left-operand digit	0	$\{0\}$	$e : (1, e, \mathsf{R})$
1	Find the input separator	0	$\{0\}$	$e : (1, e, \mathsf{R})$ $+ : (2, +, \mathsf{L})$
2	Descend below units digit	1	$\{1\}$	$e : (3, e, \mathsf{D})$
3	Place the working marker	1	$\{2\}$	$\lambda : (4, +, \mathsf{U})$
4	Ensure a digit above marker	16	$\{3\}$	$e : (6, e, \mathsf{R})$ $\lambda : (5, 0, \mathsf{R})$
5	Adjacent existing digit	2	$\{4\}$	$e : (6, e, \mathsf{R})$
6	Scan accumulator to separator	2	$\{4\}$	$e : (6, e, \mathsf{R})$ $+ : (7, +, \mathsf{R})$
7	Inspect remaining right operand	2	$\{5\}$	$e : (8, e, \mathsf{R})$ $\lambda : (54, \lambda, \mathsf{L})$
8	Scan remaining right operand	2	$\{5\}$	$e : (8, e, \mathsf{R})$ $\lambda : (9, \lambda, \mathsf{L})$
9	Read rightmost factor digit	3	$\{6\}$	$d : (F_d, \lambda, \mathsf{D})$ $(d \in D)$
50	Propagate decimal carry	14	$\{11\}$	$9 : (50, 0, \mathsf{L})$ $h : (51, h + 1, \mathsf{D})$ $\lambda : (51, 1, \mathsf{D})$
51	First return cell after carry	15	$\{15\}$	$\lambda : (52, \lambda, \mathsf{R})$
52	Return to working marker	15	$\{15\}$	$\lambda : (52, \lambda, \mathsf{R})$ $+ : (3, \lambda, \mathsf{L})$
53	Erase and shift marker	15	$\{11, 15\}$	$+ : (3, \lambda, \mathsf{L})$
54	Erase input separator	3	$\{6\}$	$+ : (55, \lambda, \mathsf{D})$

i	Phase	g_i	Q_i	Read : (next phase, write, move)
55	Begin cleanup below separator	17	$\{24\}$	$\lambda : (56, \lambda, \mathsf{L})$
56	Erase final working marker	17	$\{25\}$	$\lambda : (56, \lambda, \mathsf{L}) \quad + : (57, \lambda, \mathsf{U})$
57	Preserve final digit and stop	17	$\{26\}$	$e : (58, e, \mathsf{S})$
58	Halted	18	$\{51\}$	–

Why These Read Symbols Cover Every Valid Run. Both operands contain at least one digit. Initialization therefore reads a digit before the separator, descends below a digit and places the marker on a blank. Lemma H.15 places an existing digit immediately after each zero extension. The scan crosses the contiguous accumulator to the separator. Immediately to the separator's right, a blank means the next left step reads that separator; after any digit, the next left step instead reads a digit.

A carry moves strictly left of the marker, so the first return cell on row 1 contains a blank. A noncarry addition descends directly onto the marker. Erasing and shifting the marker reaches a blank. Cleanup starts below the separator on a blank, walks to the marker, then ascends to a digit. These facts establish the table's restrictions.

Digit-Transport Phases For each $d \in D$, four phases carry the selected digit to the accumulator:

$$F_d = 10 + 4d, \qquad H_d = F_d + 1, \qquad W_d = F_d + 2, \qquad A_d = F_d + 3.$$

All four phases use expert state $4 + d$, paired with singleton controller sets containing f_d, h_d, w_d, a_d, respectively. The protocol follows:

Phase	Meaning	Read : (next phase, write, move)
F_d	First transport blank	$\lambda : (H_d, \lambda, \mathsf{L})$
H_d	Second transport blank	$\lambda : (W_d, \lambda, \mathsf{L})$
W_d	Walk left to the marker	$\lambda : (W_d, \lambda, \mathsf{L}) \quad + : (A_d, +, \mathsf{U})$
A_d	Add at the marked column	$e : (53, d + e, \mathsf{D})$ if $d + e < 10$
		$e : (50, d + e - 10, \mathsf{L})$ if $d + e \geq 10$

The Actual Controller-State Numbers. This table expands the entire 40-phase family.

d	Expert $4 + d$	f_d	h_d	w_d	a_d
0	4	27	28	29	30
1	5	12	13	14	11
2	6	39	40	41	42
3	7	31	32	33	34
4	8	43	44	45	46
5	9	47	48	49	50
6	10	7	8	9	10
7	11	20	21	22	23
8	12	16	17	18	19
9	13	35	36	37	38

Transport Coverage. Equation (19) gives $r_k - p_k = n + 1 \geq 2$. The first two transport cells therefore contain blanks. After two left steps, the head reaches either the marker or a blank to the marker's right. The walk reaches the marker and ascends to a digit ensured at the start of the round. The two branches implement decimal addition with at most one carry:

$$d + e = s + 10c, \qquad s \in D, \quad c \in \{0, 1\}.$$

This accounts for every pair (d, e), including 0, 9 and all carry cases.

Multiple Roles of Controller State 11. State 11 performs addition of 1 when reading a digit, propagates carries when entered to the left of an overflowing column, and erases the marker when reading $+$. With $d = 1$ and no carry, the controller enters state 15; every other noncarry branch enters state 11. Both belong to $Q_{53} = \{11, 15\}$ and have the same required action on $+$. Every overflow

enters state $11 \in Q_{50}$. The phase relation distinguishes these geometric contexts and therefore does not require a one-to-one correspondence between controller states.

Local Transition Identity. For every phase i, controller state $q \in Q_i$ and listed branch $x : (j, w, M)$, direct comparison with the transition maps gives

$$\delta_{\mathcal{E}}(g_i, x) = (g_j, w, M), \qquad \delta_{\mathcal{C}}(q, x) = (q', w, M), \quad q' \in Q_j. \tag{20}$$

Every required entry exists. A transition reaches a halt state exactly when $j = 58$, with $M = S$. The certificate contains 60 phase–state pairs, including the terminal pair, and 266 local checks of (20). The checks cover all 223 defined nonterminal controller entries. The remaining 12 entries contain halt loops, excluded from PTOs.

Proof of Theorem H.14. The accumulator invariant and the ten digit-transport families cover every possible decimal digit pair and carry. Every valid execution follows the 59-phase protocol and terminates. The state relation satisfies all 266 instances of (11), including the initial pair and final stopping action. Thus Lemma H.2 proves equality of the complete PTOs and final configurations. □

H.7 Decimal Multiplication

Let $D = \{0, \dots, 9\}$ and consider $a\,x\,b$, with $a \in D^m$, $b \in D^n$, $m, n \geq 1$. Write $A = \mathrm{val}_{10}(a)$ and $B = \mathrm{val}_{10}(b)$. The expert and metric-extracted controller start in state 0 and halt in states 107 and 181, respectively.

Theorem H.18 (Exact Decimal Multiplication)**.** *The metric-extracted controller produces the expert PTO on every valid decimal-multiplication input. Both executions terminate with the $(m + n)$-digit decimal representation of AB, including leading-zero padding, on row $n - 1$ in columns $-n, \dots, m - 1$. The final head stops at $(-n - 1, n - 1)$.*

Copying and Partial Products Write $b = b_{n-1} \cdots b_0$, so $B = \sum_{i=0}^{n-1} b_i 10^i$. Assign global weight 10^{m-1-z} to column z. Column $m - 1$ carries the units digit of the final product.

Lemma H.19 (Copy Staircase)**.** *Before product formation, the expert constructs exactly n occupied rows. For $0 \leq i < n$, row i contains*

$$a\,x\,b_{n-1} \cdots b_i \quad \textit{starting in column } -i. \tag{21}$$

All remaining cells contain blanks.

Proof. Row 0 contains the input. On row i, the expert scans to the right blank, steps left over b_i, and inspects the preceding cell. A digit at this position implies at least two remaining factor digits. The expert copies every symbol except b_i one row down and one column left, from right to left. Each destination contains a blank before writing; the source row remains unchanged.

The first descent reaches a blank. Subsequent descents reach the copied right neighbor: a digit, except when copying the last digit of a, where the lower cell contains the copied x. The descent from the source x reaches a digit. After copying the first digit of a, the expert reads the left blank and descends onto the first digit of the new row. The new row satisfies (21), with one fewer factor digit.

When one factor digit remains, the second left step reaches x and starts product formation. Thus $n - 1$ copies produce exactly the stated staircase. □

Lemma H.20 (Aligned Partial Products)**.** *After product formation, row i contains the $(m + 1)$-digit representation of Ab_i on columns $-i - 1, \dots, m - i - 1$. The leading digit belongs to $\{0, \dots, 8\}$. In global column weights, this row represents $Ab_i 10^i$. All remaining cells contain blanks.*

Proof. The expert processes rows from $n - 1$ upwards. On row i, the expert erases the selected factor $f = b_i$, the remaining factor prefix, and x, then walks left across a. For $f = 0$, each digit becomes 0, followed by one padding 0 on the left blank. For $1 \leq f \leq 9$, the control state stores carry c, initially 0, and applies

$$fd + c = w + 10c', \qquad w = (fd + c) \bmod 10, \quad c' = \lfloor (fd + c)/10 \rfloor. \tag{22}$$

The bound $0 \le c < f$ persists because $fd + c \le 9f + f - 1 = 10f - 1$. The usual suffix invariant for (22) gives the product digits; the left blank receives the final carry. Thus the leading digit never exceeds $f - 1 \le 8$.

The right scan ends at column $m - i$ and moves up. For $i > 0$, the upper cell contains the last digit of a, followed by x and the remaining factor prefix. The expert scans this prefix, selects the last digit, and repeats. Every higher row contains at least two factor digits, so at least one further digit follows the selected factor during leftward erasure. For $i = 0$, the upper cell contains a blank, starting summation on row 0. □

Every scan crosses a finite word. Copying shortens the factor prefix; product formation visits each occupied row once. Both passes terminate.

Accumulation and Carry Confinement

Lemma H.21 (Fixed-Width Accumulator)**.** *After merging through row k, $0 \le k < n$, all rows above k contain blanks. Row k contains a contiguous $(m + k + 1)$-digit accumulator on columns $-k - 1, \dots, m - 1$, with value*

$$S_k = A \sum_{i=0}^{k} b_i 10^i.$$

Rows below k retain the partial products from Lemma H.20. At inspection of the next row, expert state 88 has head $(-k - 2, k + 1)$.

Proof. For $k = 0$, the expert scans left across the product on row 0, reaches column -2, and descends to inspect row 1. If row $k + 1$ exists, inspection reads the fresh leading digit in $\{0, \dots, 8\}$. The expert scans the lower product to the right blank at $m - k - 1$ and moves up. The upper cell contains an accumulator digit, since

$$-k - 1 \le m - k - 1 \le m - 1.$$

The expert scans to the upper right blank and steps left to column $m - 1$. The lower product ends at $m - k - 2$, so the first lower cell contains a blank.

For upper digit u, lower digit v (a blank contributes zero), and incoming carry $c \in \{0, 1\}$, the update applies

$$u + v + c = w + 10c', \qquad w \in D, \quad c' \in \{0, 1\}. \tag{23}$$

The expert erases u, writes w below, returns to the erased upper cell, and moves left. The processed lower suffix contains the correct sum suffix; the unprocessed cells retain the old digits; c' records the contribution to the next column. Equation (23) preserves this invariant.

The lower word extends one column farther left than the upper word. After the upper word ends, carry zero preserves this lower leading digit. Carry one increments the leading digit and clears, since the digit never exceeds 8. The expert reaches column $-k - 3$ on row $k + 1$ and descends. Row k contains only blanks; row $k + 1$ represents $S_k + Ab_{k+1}10^{k+1} = S_{k+1}$ on the claimed support. □

Lemma H.22 (Boundary Restrictions)**.** *A carry-one update beneath an upper digit reads an original lower digit. A blank lower update guarantees another upper digit on the next left step. A carry beyond the upper word reads a lower digit in $\{0, \dots, 8\}$ and clears.*

Proof. The blank gap lies strictly to the right of the original lower word. The first update has carry zero, and $u + 0 \le 9$, so no carry originates inside this gap. Carry one can first arise only inside the lower digit block; further leftward updates stay inside that block until the leading digit.

A blank lower cell has column $z \ge m - k - 1$. The next upper column satisfies $z - 1 \ge m - k - 2 \ge -k - 1$, using $m \ge 1$, and therefore still contains a digit. Finally, the lower leading digit lies one column left of the upper word. Lemma H.20 bounds this untouched digit by 8; adding 1 clears any final carry without extending the fixed-width support. □

Each merge consumes finitely many upper digits and at most one extra carry column. After $n - 1$ merges, state 88 reads the blank below the final row, moves up to state 106, and emits S into halt state 107. Lemma H.21 gives the final product, padding, and head position.

Section H.7 gives 207 phases. Phase i pairs expert state g_i with controller state $q \in Q_i$, on an identical grid and head. An edge $i \xrightarrow{x} j$ permits read symbol x and advances to phase j. The protocol begins at phase 0 and ends at phase 206. All phase sets contain one controller state except $Q_{201} = \{26, 106\}$.

Lemma H.23 (Protocol Coverage)**.** *Every expert execution in the stated domain follows the protocol in Section H.7 from phase* 0 *to phase* 206*.*

Proof. The following geometric and arithmetic conditions justify every read restriction.

Copying. The initial scan and every restart encounter a nonempty left operand, then x, then a nonempty right operand before the right blank. The second left step reads a factor digit or x. Lemma H.19 distinguishes the first descent (blank), ordinary descents (digits), and the descent directly above the copied operator (x). Every destination contains a blank. After the first copied symbol, the next source symbol contains a digit or x; no left blank can occur yet. After copying x, the next source symbol contains the last digit of a. The ordinary copy loop eventually reaches the left blank and restarts.

Products. The bottom row contains one factor digit; erasing that digit immediately exposes x. Every higher row contains at least two factor digits; erasing the selected factor therefore exposes another digit before x. After erasing x, the first product update reads a digit because $m \geq 1$. Later updates read digits followed by the left blank. Equation (22) determines the next carry phase and preserves the carry bound. The completed product scan reads digits followed by a blank. Lemma H.20 places the head on an upper digit followed by x and a nonempty factor prefix, or on a blank above row 0.

Alignment and Summation. Summation starts on the right blank of row 0. The initial left scan crosses a nonempty product. Inspection below reads a fresh leading digit from $\{0, \ldots, 8\}$ or a blank below the last row. After a leading digit, at least one lower digit remains because $m \geq 1$. Lemma H.21 places the first upper alignment step on a digit; further alignment steps read digits until the right blank. The first upper units digit has carry zero and a blank lower cell.

After every lower update, the upper return cell contains the erased blank. A blank lower update leads to a nonblank upper read by Lemma H.22. An update inside the lower digit block leads to an upper digit or the left blank. Carry-one digit updates remain inside the lower digit block. At the end of the upper word, carry zero preserves the untouched lower leading digit, while carry one increments that digit and clears. Both paths reach the blank immediately left of the merged word. The final stop reads a blank.

These conditions give exactly the permitted branches used by the expert. Induction over individual moves, copied rows, and merged rows establishes coverage. The finite-pass arguments in Lemma H.19–Lemma H.21 give termination at phase 206. □

The protocol may permit additional observation sequences. Checking every permitted branch therefore suffices for all valid grid executions.

Finite Local Certificate. For every protocol edge $i \xrightarrow{x} j$ and every $q \in Q_i$, direct comparison with the two transition maps gives

$$\delta_{\mathcal{E}}(g_i, x) = (g_j, w, M), \qquad \delta_{\mathcal{C}}(q, x) = (q', w, M), \qquad q' \in Q_j. \tag{24}$$

Every required entry exists. A transition reaches halt exactly when $j = 206$, with motion S. Direct substitution verifies all 1,509 obligations across 208 phase–state pairs. These obligations cover all 1,259 defined nonterminal controller entries. The remaining 12 defined entries contain halt loops, which never contribute to a PTO.

The Expert Transition Map. The following formulas supply every action needed by the decimal certificate. Let $d, f \in \{0, \ldots, 9\}$ and $z \in \{\lambda, x, 0, \ldots, 9\}$. Define

$$P_{f,c} = 38 + \frac{f(f-1)}{2} + c \quad (1 \leq f \leq 9,\ 0 \leq c < f), \qquad V_v = \begin{cases} 92 + v, & 0 \leq v < 10, \\ 105, & v = 10. \end{cases}$$

An entry $(q, x) \mapsto (q', w, M)$ specifies an actual state transition. A symbol d in a written field denotes the corresponding decimal digit.

State	Read	(successor, write, motion)
0	d, x	$(0, \text{read}, R)$
0	λ	$(1, \lambda, L)$
1	d	$(2, d, L)$
2	d	$(3+d, d, D)$
2	x	$(26, x, R)$
$3+d$	z	$(14+d, z, L)$
13	d	$(24, d, L)$
$14+d$	λ	$(25, d, U)$
24	λ	$(25, x, U)$
25	d	$(3+d, d, D)$
25	x	$(13, x, D)$
25	λ	$(0, \lambda, D)$
26	f	$(27+f, \lambda, L)$
$27+f$	d	$(27+f, \lambda, L)$
27	x	$(37, \lambda, L)$
$27+f, f>0$	x	$(P_{f,0}, \lambda, L)$
37	d	$(37, 0, L)$
37	λ	$(83, 0, R)$
$P_{f,c}$	d	$(P_{f,c'}, w, L)$, where $fd + c = 10c' + w, 0 \le w < 10$
$P_{f,c}$	λ	$(83, c, R)$
83	d	$(83, d, R)$
83	λ	$(84, \lambda, U)$
84	d	$(85, d, R)$
84	λ	$(86, \lambda, D)$
85	d, x	$(85, \text{read}, R)$
85	λ	$(26, \lambda, L)$
86	λ	$(87, \lambda, L)$
87	d	$(87, d, L)$
87	λ	$(88, \lambda, D)$
88	d	$(89, d, R)$
88	λ	$(106, \lambda, U)$
89	d	$(89, d, R)$
89	λ	$(104, \lambda, U)$
104	d	$(104, d, R)$
104	λ	$(90, \lambda, L)$
90	d	(V_d, λ, D)
90	λ	$(87, \lambda, D)$
102	d	(V_{d+1}, λ, D)
102	λ	$(93, \lambda, D)$
91	λ	$(90, \lambda, L)$
103	λ	$(102, \lambda, L)$
V_v	d or λ	$(91+12c', w, U)$, where $v + d = 10c' + w$ and a blank contributes $d = 0$
106	z	$(107, z, S)$

The phase restrictions below exclude every omitted expert transition. In particular, a final carry only encounters a lower leading digit at most 8.

Phases and Controller-State Relation The table gives the phase index i, expert state g_i, controller set Q_i, and permitted reads with successor phase indices. For a digit vector $[j_0, \ldots, j_9]$, read d advances to j_d. The notation $D : j$ sends every decimal digit to phase j; $0{:}8 : j$ permits only digits $0, \ldots, 8$. Unlisted reads never occur in the corresponding phase of a valid input.

The expert transition supplies the write and motion for each edge; Equation (24) checks both transition maps against that action. The copying moves follow Lemma H.19; product updates follow (22); addition updates follow (23). Phase labels describe positions before the next action.

i	Phase	g_i	Q_i	Read : next phase
0	First left-operand digit	0	$\{0\}$	$D : 1$
1	Scan left operand	0	$\{0\}$	$D : 1 \quad x : 2$
2	First factor digit	0	$\{1\}$	$D : 3$
3	Scan right operand	0	$\{1\}$	$D : 3 \quad \lambda : 4$
4	Skip last factor digit	1	$\{2\}$	$D : 5$
5	Copy or start products	2	$\{3\}$	$D : [6, 8, 10, 12, 14, 16, 18, 20, 22, 24] \quad x : 62$
6	First descent, digit 0	3	$\{172\}$	$\lambda : 7$
7	First copied write, digit 0	14	$\{173\}$	$\lambda : 26$
8	First descent, digit 1	4	$\{123\}$	$\lambda : 9$
9	First copied write, digit 1	15	$\{124\}$	$\lambda : 26$
10	First descent, digit 2	5	$\{64\}$	$\lambda : 11$
11	First copied write, digit 2	16	$\{65\}$	$\lambda : 26$
12	First descent, digit 3	6	$\{4\}$	$\lambda : 13$
13	First copied write, digit 3	17	$\{5\}$	$\lambda : 26$
14	First descent, digit 4	7	$\{87\}$	$\lambda : 15$
15	First copied write, digit 4	18	$\{88\}$	$\lambda : 26$
16	First descent, digit 5	8	$\{39\}$	$\lambda : 17$
17	First copied write, digit 5	19	$\{40\}$	$\lambda : 26$
18	First descent, digit 6	9	$\{163\}$	$\lambda : 19$
19	First copied write, digit 6	20	$\{164\}$	$\lambda : 26$
20	First descent, digit 7	10	$\{51\}$	$\lambda : 21$
21	First copied write, digit 7	21	$\{52\}$	$\lambda : 26$
22	First descent, digit 8	11	$\{128\}$	$\lambda : 23$
23	First copied write, digit 8	22	$\{129\}$	$\lambda : 26$
24	First descent, digit 9	12	$\{79\}$	$\lambda : 25$
25	First copied write, digit 9	23	$\{80\}$	$\lambda : 26$
26	Source after first copy	25	$\{6\}$	$D : [27, 29, 31, 33, 35, 37, 39, 41, 43, 45] \quad x : 48$
27	Ordinary descent, digit 0	3	$\{178\}$	$D : 28$
28	Ordinary copied write, digit 0	14	$\{159\}$	$\lambda : 47$
29	Ordinary descent, digit 1	4	$\{134\}$	$D : 30$
30	Ordinary copied write, digit 1	15	$\{135\}$	$\lambda : 47$
31	Ordinary descent, digit 2	5	$\{68\}$	$D : 32$
32	Ordinary copied write, digit 2	16	$\{67\}$	$\lambda : 47$
33	Ordinary descent, digit 3	6	$\{125\}$	$D : 34$
34	Ordinary copied write, digit 3	17	$\{11\}$	$\lambda : 47$
35	Ordinary descent, digit 4	7	$\{166\}$	$D : 36$
36	Ordinary copied write, digit 4	18	$\{90\}$	$\lambda : 47$
37	Ordinary descent, digit 5	8	$\{165\}$	$D : 38$
38	Ordinary copied write, digit 5	19	$\{131\}$	$\lambda : 47$
39	Ordinary descent, digit 6	9	$\{55\}$	$D : 40$
40	Ordinary copied write, digit 6	20	$\{42\}$	$\lambda : 47$
41	Ordinary descent, digit 7	10	$\{170\}$	$D : 42$
42	Ordinary copied write, digit 7	21	$\{148\}$	$\lambda : 47$
43	Ordinary descent, digit 8	11	$\{150\}$	$D : 44$
44	Ordinary copied write, digit 8	22	$\{116\}$	$\lambda : 47$
45	Ordinary descent, digit 9	12	$\{167\}$	$D : 46$
46	Ordinary copied write, digit 9	23	$\{54\}$	$\lambda : 47$
47	Continue source copy	25	$\{12\}$	$D : [27, 29, 31, 33, 35, 37, 39, 41, 43, 45] \quad x : 48 \quad \lambda : 61$
48	Descend below operator	13	$\{7\}$	$D : 49$
49	Write copied operator	24	$\{8\}$	$\lambda : 50$
50	Copy last digit of a	25	$\{9\}$	$D : [51, 52, 53, 54, 55, 56, 57, 58, 59, 60]$
51	Descend above x, digit 0	3	$\{158\}$	$x : 28$
52	Descend above x, digit 1	4	$\{145\}$	$x : 30$
53	Descend above x, digit 2	5	$\{66\}$	$x : 32$
54	Descend above x, digit 3	6	$\{10\}$	$x : 34$
55	Descend above x, digit 4	7	$\{89\}$	$x : 36$
56	Descend above x, digit 5	8	$\{130\}$	$x : 38$
57	Descend above x, digit 6	9	$\{41\}$	$x : 40$

Continued on the next page

Simulation Certificate (Continued)

i	Phase	g_i	Q_i	Read : next phase
58	Descend above x, digit 7	10	$\{147\}$	$x : 42$
59	Descend above x, digit 8	11	$\{115\}$	$x : 44$
60	Descend above x, digit 9	12	$\{53\}$	$x : 46$
61	Restart on copied row	0	$\{13\}$	$D : 1$
62	Bottom-row factor	26	$\{14\}$	$D : [63, 66, 69, 73, 78, 84, 91, 99, 108, 118]$
63	Erase x, bottom factor 0	27	$\{111\}$	$x : 64$
64	First product digit, factor 0	37	$\{62\}$	$D : 65$
65	Continue product by zero	37	$\{62\}$	$D : 65 \quad \lambda : 129$
66	Erase x, bottom factor 1	28	$\{78\}$	$x : 67$
67	First product digit, factor 1	38	$\{24\}$	$D : 68$
68	Product by 1, carry 0	38	$\{24\}$	$D : 68 \quad \lambda : 129$
69	Erase x, bottom factor 2	29	$\{69\}$	$x : 70$
70	First product digit, factor 2	39	$\{70\}$	$D : [71, 71, 71, 71, 71, 72, 72, 72, 72, 72]$
71	Product by 2, carry 0	39	$\{70\}$	$D : [71, 71, 71, 71, 71, 72, 72, 72, 72, 72] \quad \lambda : 129$
72	Product by 2, carry 1	40	$\{143\}$	$D : [71, 71, 71, 71, 71, 72, 72, 72, 72, 72] \quad \lambda : 129$
73	Erase x, bottom factor 3	30	$\{15\}$	$x : 74$
74	First product digit, factor 3	41	$\{16\}$	$D : [75, 75, 75, 75, 76, 76, 76, 77, 77, 77]$
75	Product by 3, carry 0	41	$\{16\}$	$D : [75, 75, 75, 75, 76, 76, 76, 77, 77, 77] \quad \lambda : 129$
76	Product by 3, carry 1	42	$\{48\}$	$D : [75, 75, 75, 76, 76, 76, 76, 77, 77, 77] \quad \lambda : 129$
77	Product by 3, carry 2	43	$\{155\}$	$D : [75, 75, 75, 76, 76, 76, 77, 77, 77, 77] \quad \lambda : 129$
78	Erase x, bottom factor 4	31	$\{75\}$	$x : 79$
79	First product digit, factor 4	44	$\{76\}$	$D : [80, 80, 80, 81, 81, 82, 82, 82, 83, 83]$
80	Product by 4, carry 0	44	$\{76\}$	$D : [80, 80, 80, 81, 81, 82, 82, 82, 83, 83] \quad \lambda : 129$
81	Product by 4, carry 1	45	$\{91\}$	$D : [80, 80, 80, 81, 81, 82, 82, 82, 83, 83] \quad \lambda : 129$
82	Product by 4, carry 2	46	$\{77\}$	$D : [80, 80, 81, 81, 81, 82, 82, 83, 83, 83] \quad \lambda : 129$
83	Product by 4, carry 3	47	$\{117\}$	$D : [80, 80, 81, 81, 81, 82, 82, 83, 83, 83] \quad \lambda : 129$
84	Erase x, bottom factor 5	32	$\{43\}$	$x : 85$
85	First product digit, factor 5	48	$\{44\}$	$D : [86, 86, 87, 87, 88, 88, 89, 89, 90, 90]$
86	Product by 5, carry 0	48	$\{44\}$	$D : [86, 86, 87, 87, 88, 88, 89, 89, 90, 90] \quad \lambda : 129$
87	Product by 5, carry 1	49	$\{153\}$	$D : [86, 86, 87, 87, 88, 88, 89, 89, 90, 90] \quad \lambda : 129$
88	Product by 5, carry 2	50	$\{144\}$	$D : [86, 86, 87, 87, 88, 88, 89, 89, 90, 90] \quad \lambda : 129$
89	Product by 5, carry 3	51	$\{45\}$	$D : [86, 86, 87, 87, 88, 88, 89, 89, 90, 90] \quad \lambda : 129$
90	Product by 5, carry 4	52	$\{95\}$	$D : [86, 86, 87, 87, 88, 88, 89, 89, 90, 90] \quad \lambda : 129$
91	Erase x, bottom factor 6	33	$\{107\}$	$x : 92$
92	First product digit, factor 6	53	$\{108\}$	$D : [93, 93, 94, 94, 95, 96, 96, 97, 97, 98]$
93	Product by 6, carry 0	53	$\{108\}$	$D : [93, 93, 94, 94, 95, 96, 96, 97, 97, 98] \quad \lambda : 129$
94	Product by 6, carry 1	54	$\{171\}$	$D : [93, 93, 94, 94, 95, 96, 96, 97, 97, 98] \quad \lambda : 129$
95	Product by 6, carry 2	55	$\{110\}$	$D : [93, 93, 94, 95, 95, 96, 96, 97, 98, 98] \quad \lambda : 129$
96	Product by 6, carry 3	56	$\{174\}$	$D : [93, 93, 94, 95, 95, 96, 96, 97, 98, 98] \quad \lambda : 129$
97	Product by 6, carry 4	57	$\{142\}$	$D : [93, 94, 94, 95, 95, 96, 97, 97, 98, 98] \quad \lambda : 129$
98	Product by 6, carry 5	58	$\{109\}$	$D : [93, 94, 94, 95, 95, 96, 97, 97, 98, 98] \quad \lambda : 129$
99	Erase x, bottom factor 7	34	$\{56\}$	$x : 100$
100	First product digit, factor 7	59	$\{57\}$	$D : [101, 101, 102, 103, 103, 104, 105, 105, 106, 107]$
101	Product by 7, carry 0	59	$\{57\}$	$D : [101, 101, 102, 103, 103, 104, 105, 105, 106, 107] \quad \lambda : 129$
102	Product by 7, carry 1	60	$\{168\}$	$D : [101, 101, 102, 103, 103, 104, 105, 106, 106, 107] \quad \lambda : 129$
103	Product by 7, carry 2	61	$\{127\}$	$D : [101, 101, 102, 103, 104, 104, 105, 106, 106, 107] \quad \lambda : 129$
104	Product by 7, carry 3	62	$\{179\}$	$D : [101, 102, 102, 103, 104, 104, 105, 106, 106, 107] \quad \lambda : 129$
105	Product by 7, carry 4	63	$\{59\}$	$D : [101, 102, 102, 103, 104, 104, 105, 106, 107, 107] \quad \lambda : 129$
106	Product by 7, carry 5	64	$\{169\}$	$D : [101, 102, 102, 103, 104, 105, 105, 106, 107, 107] \quad \lambda : 129$
107	Product by 7, carry 6	65	$\{58\}$	$D : [101, 102, 103, 103, 104, 105, 105, 106, 107, 107] \quad \lambda : 129$
108	Erase x, bottom factor 8	35	$\{112\}$	$x : 109$
109	First product digit, factor 8	66	$\{113\}$	$D : [110, 110, 111, 112, 113, 114, 114, 115, 116, 117]$
110	Product by 8, carry 0	66	$\{113\}$	$D : [110, 110, 111, 112, 113, 114, 114, 115, 116, 117] \quad \lambda : 129$
111	Product by 8, carry 1	67	$\{137\}$	$D : [110, 110, 111, 112, 113, 114, 114, 115, 116, 117] \quad \lambda : 129$
112	Product by 8, carry 2	68	$\{121\}$	$D : [110, 111, 111, 112, 113, 114, 115, 115, 116, 117] \quad \lambda : 129$
113	Product by 8, carry 3	69	$\{114\}$	$D : [110, 111, 111, 112, 113, 114, 115, 115, 116, 117] \quad \lambda : 129$

Continued on the next page

Simulation Certificate (Continued)

i	Phase	g_i	Q_i	Read : next phase
114	Product by 8, carry 4	70	$\{132\}$	$D : [110, 111, 112, 112, 113, 114, 115, 116, 116, 117]$ $\lambda : 129$
115	Product by 8, carry 5	71	$\{149\}$	$D : [110, 111, 112, 112, 113, 114, 115, 116, 116, 117]$ $\lambda : 129$
116	Product by 8, carry 6	72	$\{120\}$	$D : [110, 111, 112, 113, 113, 114, 115, 116, 117, 117]$ $\lambda : 129$
117	Product by 8, carry 7	73	$\{136\}$	$D : [110, 111, 112, 113, 113, 114, 115, 116, 117, 117]$ $\lambda : 129$
118	Erase x, bottom factor 9	36	$\{81\}$	$x : 119$
119	First product digit, factor 9	74	$\{82\}$	$D : [120, 120, 121, 122, 123, 124, 125, 126, 127, 128]$
120	Product by 9, carry 0	74	$\{82\}$	$D : [120, 120, 121, 122, 123, 124, 125, 126, 127, 128]$ $\lambda : 129$
121	Product by 9, carry 1	75	$\{151\}$	$D : [120, 121, 121, 122, 123, 124, 125, 126, 127, 128]$ $\lambda : 129$
122	Product by 9, carry 2	76	$\{177\}$	$D : [120, 121, 122, 122, 123, 124, 125, 126, 127, 128]$ $\lambda : 129$
123	Product by 9, carry 3	77	$\{154\}$	$D : [120, 121, 122, 123, 123, 124, 125, 126, 127, 128]$ $\lambda : 129$
124	Product by 9, carry 4	78	$\{176\}$	$D : [120, 121, 122, 123, 124, 124, 125, 126, 127, 128]$ $\lambda : 129$
125	Product by 9, carry 5	79	$\{83\}$	$D : [120, 121, 122, 123, 124, 125, 125, 126, 127, 128]$ $\lambda : 129$
126	Product by 9, carry 6	80	$\{156\}$	$D : [120, 121, 122, 123, 124, 125, 126, 126, 127, 128]$ $\lambda : 129$
127	Product by 9, carry 7	81	$\{152\}$	$D : [120, 121, 122, 123, 124, 125, 126, 127, 127, 128]$ $\lambda : 129$
128	Product by 9, carry 8	82	$\{98\}$	$D : [120, 121, 122, 123, 124, 125, 126, 127, 128, 128]$ $\lambda : 129$
129	Scan completed product	83	$\{17\}$	$D : 129$ $\lambda : 130$
130	Inspect row above	84	$\{18\}$	$D : 131$ $\lambda : 155$
131	Cross upper operator	85	$\{19\}$	$x : 132$
132	First upper factor digit	85	$\{20\}$	$D : 133$
133	Scan upper factor prefix	85	$\{20\}$	$D : 133$ $\lambda : 134$
134	Select upper factor	26	$\{21\}$	$D : [135, 137, 139, 141, 143, 145, 147, 149, 151, 153]$
135	First prefix erasure, factor 0	27	$\{60\}$	$D : 136$
136	Later prefix erasure, factor 0	27	$\{61\}$	$D : 136$ $x : 64$
137	First prefix erasure, factor 1	28	$\{22\}$	$D : 138$
138	Later prefix erasure, factor 1	28	$\{23\}$	$D : 138$ $x : 67$
139	First prefix erasure, factor 2	29	$\{71\}$	$D : 140$
140	Later prefix erasure, factor 2	29	$\{72\}$	$D : 140$ $x : 70$
141	First prefix erasure, factor 3	30	$\{46\}$	$D : 142$
142	Later prefix erasure, factor 3	30	$\{47\}$	$D : 142$ $x : 74$
143	First prefix erasure, factor 4	31	$\{92\}$	$D : 144$
144	Later prefix erasure, factor 4	31	$\{93\}$	$D : 144$ $x : 79$
145	First prefix erasure, factor 5	32	$\{84\}$	$D : 146$
146	Later prefix erasure, factor 5	32	$\{85\}$	$D : 146$ $x : 85$
147	First prefix erasure, factor 6	33	$\{140\}$	$D : 148$
148	Later prefix erasure, factor 6	33	$\{141\}$	$D : 148$ $x : 92$
149	First prefix erasure, factor 7	34	$\{160\}$	$D : 150$
150	Later prefix erasure, factor 7	34	$\{161\}$	$D : 150$ $x : 100$
151	First prefix erasure, factor 8	35	$\{118\}$	$D : 152$
152	Later prefix erasure, factor 8	35	$\{119\}$	$D : 152$ $x : 109$
153	First prefix erasure, factor 9	36	$\{96\}$	$D : 154$
154	Later prefix erasure, factor 9	36	$\{97\}$	$D : 154$ $x : 119$
155	Start summation	86	$\{25\}$	$\lambda : 156$
156	First initial left digit	87	$\{26\}$	$D : 157$
157	Continue initial left scan	87	$\{26\}$	$D : 157$ $\lambda : 158$
158	Inspect fresh lower row	88	$\{27\}$	$0{:}8 : 159$ $\lambda : 205$
159	First digit after leading cell	89	$\{28\}$	$D : 160$
160	Scan lower product	89	$\{28\}$	$D : 160$ $\lambda : 161$
161	First upper alignment digit	104	$\{29\}$	$D : 162$
162	Continue upper alignment	104	$\{30\}$	$D : 162$ $\lambda : 163$
163	Read upper units digit	90	$\{31\}$	$D : [164, 165, 166, 167, 168, 169, 170, 171, 172, 173]$
164	First lower update, value 0	92	$\{63\}$	$\lambda : 174$
165	First lower update, value 1	93	$\{99\}$	$\lambda : 174$
166	First lower update, value 2	94	$\{138\}$	$\lambda : 174$
167	First lower update, value 3	95	$\{32\}$	$\lambda : 174$
168	First lower update, value 4	96	$\{73\}$	$\lambda : 174$
169	First lower update, value 5	97	$\{133\}$	$\lambda : 174$

Continued on the next page

Simulation Certificate (Continued)

i	Phase	g_i	Q_i	Read : next phase	
170	First lower update, value 6	98	$\{94\}$	$\lambda : 174$	
171	First lower update, value 7	99	$\{162\}$	$\lambda : 174$	
172	First lower update, value 8	100	$\{49\}$	$\lambda : 174$	
173	First lower update, value 9	101	$\{146\}$	$\lambda : 174$	
174	Return after lower blank	91	$\{33\}$	$\lambda : 175$	
175	Read next upper digit	90	$\{34\}$	$D : [178, 179, 180, 181, 182, 183, 184, 185, 186, 187]$	
176	Return with carry zero	91	$\{36\}$	$\lambda : 177$	
177	Read upper, carry zero	90	$\{37\}$	$D : [178, 179, 180, 181, 182, 183, 184, 185, 186, 187]$	$\lambda : 200$
178	Add 0 below	92	$\{35\}$	$D : 176 \quad \lambda : 174$	
179	Add 1 below	93	$\{50\}$	$D : [176, 176, 176, 176, 176, 176, 176, 176, 176, 188]$	$\lambda : 174$
180	Add 2 below	94	$\{122\}$	$D : [176, 176, 176, 176, 176, 176, 176, 176, 188, 188]$	$\lambda : 174$
181	Add 3 below	95	$\{86\}$	$D : [176, 176, 176, 176, 176, 176, 176, 188, 188, 188]$	$\lambda : 174$
182	Add 4 below	96	$\{74\}$	$D : [176, 176, 176, 176, 176, 176, 188, 188, 188, 188]$	$\lambda : 174$
183	Add 5 below	97	$\{139\}$	$D : [176, 176, 176, 176, 176, 188, 188, 188, 188, 188]$	$\lambda : 174$
184	Add 6 below	98	$\{157\}$	$D : [176, 176, 176, 176, 188, 188, 188, 188, 188, 188]$	$\lambda : 174$
185	Add 7 below	99	$\{175\}$	$D : [176, 176, 176, 188, 188, 188, 188, 188, 188, 188]$	$\lambda : 174$
186	Add 8 below	100	$\{100\}$	$D : [176, 176, 188, 188, 188, 188, 188, 188, 188, 188]$	$\lambda : 174$
187	Add 9 below	101	$\{126\}$	$D : [176, 188, 188, 188, 188, 188, 188, 188, 188, 188]$	$\lambda : 174$
188	Return with carry one	103	$\{101\}$	$\lambda : 189$	
189	Read upper, carry one	102	$\{102\}$	$D : [190, 191, 192, 193, 194, 195, 196, 197, 198, 199]$	$\lambda : 202$
190	Add 1 inside lower block	93	$\{50\}$	$D : [176, 176, 176, 176, 176, 176, 176, 176, 176, 188]$	
191	Add 2 inside lower block	94	$\{122\}$	$D : [176, 176, 176, 176, 176, 176, 176, 176, 188, 188]$	
192	Add 3 inside lower block	95	$\{86\}$	$D : [176, 176, 176, 176, 176, 176, 176, 188, 188, 188]$	
193	Add 4 inside lower block	96	$\{74\}$	$D : [176, 176, 176, 176, 176, 176, 188, 188, 188, 188]$	
194	Add 5 inside lower block	97	$\{139\}$	$D : [176, 176, 176, 176, 176, 188, 188, 188, 188, 188]$	
195	Add 6 inside lower block	98	$\{157\}$	$D : [176, 176, 176, 176, 188, 188, 188, 188, 188, 188]$	
196	Add 7 inside lower block	99	$\{175\}$	$D : [176, 176, 176, 188, 188, 188, 188, 188, 188, 188]$	
197	Add 8 inside lower block	100	$\{100\}$	$D : [176, 176, 188, 188, 188, 188, 188, 188, 188, 188]$	
198	Add 9 inside lower block	101	$\{126\}$	$D : [176, 188, 188, 188, 188, 188, 188, 188, 188, 188]$	
199	Add 10 inside lower block	105	$\{180\}$	$D : 188$	
200	Preserve leading digit	87	$\{26\}$	$0{:}8 : 201$	
201	Blank left of merged row	87	$\{26, 106\}$	$\lambda : 158$	
202	Increment leading digit	93	$\{103\}$	$0{:}8 : 203$	
203	Return after final carry	91	$\{104\}$	$\lambda : 204$	
204	Step down left of sum	90	$\{105\}$	$\lambda : 201$	
205	Emit final stop	106	$\{38\}$	$\lambda : 206$	
206	Halted	107	$\{181\}$	–	

Proof of Theorem H.18. The staircase, partial-product, accumulator, and boundary invariants establish coverage of the 207-phase protocol for arbitrary operand lengths. The displayed state relation contains 208 phase–state pairs and satisfies 1,509 local identities. The protocol excludes every undefined controller transition and terminates after the common final action. Apply Lemma H.2; the accumulator invariant supplies the final padded product and head position. □

H.8 Universal Computation: Two Certificates for One Interpreter

A simulated Turing machine has states $\{0, \ldots, N-1\}$, $N \geq 1$, tape alphabet $\Gamma = \{\lambda, 0, 1\}$, and a total transition function

$$\delta : \{0, \ldots, N-1\} \times \Gamma \longrightarrow \{0, \ldots, N-1\} \times \Gamma \times \{L, R, S\}.$$

The simulated machine starts in state 0; motion S stops after writing. The encoding below admits arbitrary program sizes. Both interpreter controllers start in state 0; the expert, metric controller, and annealed controller have halt states 14, 43, and 27, respectively.

Theorem H.24 (Exact Universal Interpretation)**.** *For every correctly encoded machine and every nonempty binary input, both extracted controllers produce the expert PTO. The annealed controller additionally covers empty binary input. At every step, the visible grids, heads, and rune records agree,*

including symbols hidden beneath runes. Both controllers and the expert halt simultaneously, or all corresponding finite execution prefixes agree forever.

H.8.1 Shared Encoding and Expert Invariants

Encoding and Rune Semantics Let $\kappa = \max(1, \lceil \log_3 N \rceil)$. Encode each state with a fixed-width, most-significant-first ternary word using L, R, S for $0, 1, 2$ respectively. Write this word as $c(q)$. The trit symbol S encodes the value 2; the motion S stops execution.

Order tape symbols as $\sigma_0 = \lambda < \sigma_1 = 0 < \sigma_2 = 1$. For $\delta(q, \sigma_i) = (q', w, M)$, program row $r = 3q + i$ contains

$$\big[\sigma_i, \lambda, c(q), 0, c(q'), 0, w, M\big]. \tag{25}$$

The source and successor codes each occupy κ separate cells. Thus the program has $3N$ rows, ordered by state and then by read symbol.

Columns	Field	Purpose
0	σ_i	Simulated tape read symbol
1	λ	Bookmark column for rune X
$2, \ldots, \kappa + 1$	$c(q)$	Source-state code
$\kappa + 2$	0	First delimiter
$\kappa + 3, \ldots, 2\kappa + 2$	$c(q')$	Successor-state code
$2\kappa + 3$	0	Second delimiter
$2\kappa + 4$	w	Symbol to write
$2\kappa + 5$	M	Simulated move L, R or S

Initial Placement. Row r occupies grid height $y = -r$, so D advances down the table and U moves up the table. Initially X covers the blank at $(1, 0)$; no cell contains Y or Z. The input occupies columns $1, \ldots, |v|$ of row $3N$, at height $-3N$. The controller starts at $(1, -3N)$, including for empty input. The tape row extends infinitely in both horizontal directions.

Rune Semantics. Each of X, Y, Z has one position and a hidden symbol. Writing a rune first restores the hidden symbol at the old rune position, then installs the rune over the symbol at the new position. Rewriting a rune in place preserves the hidden symbol. Writing an ordinary symbol at a rune removes that rune. A motion X, Y or Z jumps to the corresponding rune after the write. Ordinary motions move one cell; S stops without moving.

The *logical program* means the program with every rune replaced by the corresponding hidden symbol. Correctness depends on preserving this logical program, not on treating a rune as a destructive overwrite.

Endpoint and Prefix Invariant

Lemma H.25 (Endpoint and Prefix Invariant)**.** *At the start of every simulated step, the expert occupies state* 0 *at the simulated tape head. Rune* X *occupies column* 1 *on the first or last row of the current simulated state's three-row block. After matching* j *successor trits,* X *occupies column* $1 + j$ *on a row with the correct length-*j *successor prefix. The row index remains congruent to* 0 *or* 2 *modulo* 3*. The logical program remains unchanged.*

Proof. Initially X occupies the first row of state 0. Assume the invariant at the start of a simulated step. The expert reads the tape symbol, installs Z, jumps to X, and steps left. The symbol search selects the matching row within the current state's three-row block. The following right scan crosses column 1 and the source code to the first delimiter, leaving X in place.

For successor trit t, the expert places Y on the trit, jumps to X, and steps right to the next source-code column. All rows sharing the matched prefix form one interval. Within that interval, next-trit values increase monotonically down the table. The successor state exists, so a nonempty subinterval contains next trit t. Moving down on a smaller trit, or up on a larger trit, reaches t before leaving the matched-prefix interval.

An immediate match preserves the endpoint. A downward search enters the target subinterval at the first row, with read symbol λ; an upward search enters at the last row, with read symbol 1. Both

rows occupy endpoints of three-row state blocks. Writing X advances the matched prefix by one trit. Jumping to Y resumes instruction decoding.

After all κ trits, X occupies a row in the successor state's block. The second delimiter triggers a leftward rewind of X to column 1 on the same row. The expert then applies the prescribed write and motion at Z. Rune relocation restores every covered program symbol, preserving the logical program. Each nonstopping motion establishes the next simulated step. □

Lemma H.26 (No Missing Middle Trit)**.** *Within a fixed source-code prefix block, a vertical search cannot pass directly between an L region and an S region. A nonempty R region separates the two regions.*

Proof. The state numbers fill the contiguous interval $0, \ldots, N-1$, with a common encoding width. A prefix block contains a possibly truncated interval of consecutive integers. A block containing next-trit values L and S also contains every state in the intervening R subinterval. Each state contributes all three read-symbol rows. □

Corollary H.27 (Faithful Expert Simulation)**.** *Each simulated TM transition takes finitely many controller steps and performs the write and motion prescribed by δ. The expert halts exactly when the simulated TM issues S.*

Proof. Lemma H.25 selects the correct instruction and successor. Every scan stays within a finite program interval, and every trit search reaches a nonempty target interval. The final two instruction fields supply the prescribed write and motion at Z. Only a simulated S triggers a stopping transition. □

H.8.2 Metric-Extraction Certificate: Nonempty Inputs

The protocol records finite geometric context. Let $e \in \{0, 2\}$ identify the endpoint row occupied by X. Use $r, j \in \{0, 1, 2\}$ to index tape symbols and $t, a \in \{L, R, S\}$ to denote trits. A vertical search records the last observed trit. Each phase and parameter tuple determines an expert state.

Phase	Expert state	Allowed reading and continuation
Initial	0	Read 0 or 1; select the matching row from endpoint 0.
Dispatch(e)	0	Read σ_r; jump to selection of r.
Selection jump(e, r)	$1 + r$	Read X; step left to row index $j = e$.
Select(e, r, j)	$1 + r$	Read σ_j; if $j = r$, cross column 1; otherwise change j by $\operatorname{sgn}(r - j)$.
Cross(e, r)	4	Read X if $e = r$, and λ otherwise; enter first source trit.
First source trit(e)	4	Read any trit; enter remaining source-code scan.
Source scan(e)	4	Trit: repeat. Delimiter 0: enter first successor trit.
First successor trit(e)	5	Read a trit t; jump to X for matching t.
Successor scan(e)	5	Trit t: match t. Delimiter 0: start rewind.
Match jump(e, t)	g_t	Read X; step right and begin comparison.
First comparison(e, t)	g_t	Read a. If $a = t$, resume with endpoint e; if $a < t$, search down; if $a > t$, search up.
Search down(t, a)	g_t	Read a or the immediate successor of a. Repeat until t; then resume with endpoint 0.
Search up(t, a)	g_t	Read a or the immediate predecessor of a. Repeat until t; then resume with endpoint 2.
Resume(e, t)	g_t	Read Y; advance to successor scan with endpoint e.
Rewind jump(e)	9	Read X; step left.
Rewind(e)	9	Trit: continue left. Blank: move X here and jump to Y.
Rewind return(e)	9	Read Y; step right to the write field.
Read write(e)	10	Read σ_r; remember this write symbol.
Read motion(e, r)	$11 + r$	Read $M \in \{L, R, S\}$; jump to Z.
Apply(e, r, M)	$h_{r,M}$	Read Z; write σ_r and move M. Dispatch(e) if $M \neq S$; otherwise halt.
Halt	14	No outgoing action.

Here $g_L = 6$, $g_R = 7$, $g_S = 8$, and

$$h_{r,L} = 1 + 2r, \qquad h_{r,R} = 2 + 2r, \qquad h_{r,S} = 11 + r.$$

The protocol starts at Initial; later simulated steps start at Dispatch(e). Searches update the remembered trit after each change. Thus a downward search for S that has seen L must see R before S. The protocol permits unbounded repetition while retaining finite annotations.

Lemma H.28 (Protocol Coverage)**.** *Every expert execution on a correct encoding with nonempty input follows this protocol.*

Proof. Initially the tape head reads 0 or 1, and X occupies endpoint 0. Endpoint selection and the following column-1 reading follow from Lemma H.25. Both state codes have positive width. Prefix-block monotonicity and Lemma H.26 give the allowed vertical-search readings and endpoint updates. Rewind reaches the blank in column 1, then Y, the write field, the motion field and Z. These exhaust the phases of one simulated step; induction gives coverage for every finite prefix of the execution. □

The protocol permits additional observation sequences, including unequal lengths for the two code scans. The certificate checks every permitted transition, so this overapproximation preserves the equivalence argument.

Initial Entry, Row Selection and Code Scanning For phase i, write Q_i for the controller-state set specified in the following certificate subsections. Relate configurations with identical grids, heads and rune records, with expert state g_i and controller state in Q_i.

Initial and Subsequent Dispatches. The Initial phase has $Q_i = \{0\}$ and permits $0, 1$ only, with successors $1, 35$, respectively. Dispatch(e) has $Q_i = \{17\}$ for either endpoint e, with successors $18, 24, 26$ on $\lambda, 0, 1$. Every permitted transition writes Z and teleports to X, matching expert state 0. The certificate distinguishes the two initial selection routes.

The Complete Row-Selection Routes. Each state list gives the controller state *before each reading*; after the final reading, the controller enters state 5 at the first source-code trit. The table represents each endpoint by the corresponding read symbol, λ or 1.

Entry	Endpoint	Wanted	Readings in order	States in order
Initial	λ	0	$X, \lambda, 0, \lambda$	$1, 2, 3, 4$
Initial	λ	1	$X, \lambda, 0, 1, \lambda$	$35, 36, 37, 38, 4$
Later	λ	λ	X, λ, X	$18, 19, 20$
Later	λ	0	$X, \lambda, 0, \lambda$	$24, 25, 3, 4$
Later	λ	1	$X, \lambda, 0, 1, \lambda$	$26, 27, 37, 38, 4$
Later	1	λ	$X, 1, 0, \lambda, \lambda$	$18, 19, 32, 33, 4$
Later	1	0	$X, 1, 0, \lambda$	$24, 25, 3, 4$
Later	1	1	$X, 1, X$	$26, 27, 20$

The expert uses state $1 + r$ during selection of σ_r, then state 4 for the final column-1 reading. The X step moves left; comparisons move up or down toward the wanted symbol; the matching and crossing steps move right. Every step preserves the read symbol. Substitution in the transition maps checks every route and successor state 5.

Taking the union where Initial and Later routes share a phase gives exactly the needed multivalued sets. For endpoint 0 and wanted symbol 0, the selection-jump and first-comparison sets equal $\{1, 24\}$ and $\{2, 25\}$, respectively. For endpoint 0 and wanted symbol 1, the same sets equal $\{26, 35\}$ and $\{27, 36\}$. Every other phase/parameter set contains one controller state.

Source and Successor Code Scans. The first and subsequent source-trit phases both have $Q_i = \{5\}$. State 5 preserves each trit, moves right, and remains 5. On delimiter 0, state 5 preserves 0, moves right, and enters state 6. Both the first successor-trit phase and the subsequent successor scan have $Q_i = \{6\}$, with the following transitions:

Controller state	Read L	Read R	Read S	Read 0
6	7	10	28	11

Entries list successor states; each action writes Y and teleports to X. The delimiter branch follows at least one successor trit. Expert state 5 enters $6, 7, 8$ for trits L, R, S, and 9 for the delimiter. This gives the match-jump states $7, 10, 28$ and rewind-jump state 11.

Matching Trits, Rewinding and Updating the Tape **Match One Successor Trit.** The match-jump states $7, 10, 28$ preserve X, move right and enter $8, 10, 29$, respectively, for targets L, R, S. The following table gives the comparison actions.

Target	State	Read L	Read R	Read S
L	8	$(9, X, \mathsf{Y})$	$(8, R, \mathsf{U})$	$(8, S, \mathsf{U})$
R	10	$(10, L, \mathsf{D})$	$(9, X, \mathsf{Y})$	$(10, S, \mathsf{U})$
S	29	$(29, L, \mathsf{D})$	$(29, R, \mathsf{D})$	$(9, X, \mathsf{Y})$

These rows cover every permitted first-comparison and vertical-search phase, paired with expert states $6, 7, 8$. On a match, writing X advances the bookmark and teleporting to Y returns to the instruction. Resume(e, t) has $Q_i = \{9\}$ for every endpoint and target: state 9 preserves Y, moves right, and enters state 6.

Rewind X and Reach the Write Field. All rewind phases use expert state 9 and follow this sequence:

Phase	Controller state	Reading	Transition
Rewind jump	11	X	$(12, X, \mathsf{L})$
Rewind through code	12	$t \in \{L, R, S\}$	$(12, t, \mathsf{L})$
Rewind reaches column 1	12	λ	$(13, X, \mathsf{Y})$
Rewind return	13	Y	$(14, Y, \mathsf{R})$

The final step enters expert state 10, so ReadWrite(e) has $Q_i = \{14\}$. On reading $\lambda, 0, 1$, state 14 preserves that symbol, moves right and enters $15, 21, 30$, respectively. The corresponding expert transitions enter states $11, 12, 13$.

Motion Decoding and Tape Update. Each motion-reading transition writes Y, teleports to Z, and enters the apply state below. That state reads Z, writes the row's symbol and performs exactly the chosen motion.

Write symbol	Motion-reading state	Apply L	Apply R	Apply S
λ	15	16	23	34
0	21	40	22	42
1	30	39	31	41

The L and R apply transitions return to controller state 17, paired with expert state 0 in Dispatch(e). Either motion may reach a blank tape cell; state 17 handles that case correctly. The S transitions enter controller halt state 43, paired with expert halt state 14.

Certificate Size. These certificate subsections specify every Q_i, including the four doubletons caused by initial versus later selection. The 92 reachable parameterized phases produce 96 related phase–state pairs. The 160 local checks cover all 73 defined nonterminal entries of the extracted controller. The remaining nine defined entries contain halt loops, excluded from PTOs. Every permitted successor belongs to the next phase's controller-state set.

For each permitted protocol edge $i \xrightarrow{x} j$ and each $q \in Q_i$, the preceding certificate establishes

$$\delta_{\mathcal{E}}(g_i, x) = (g_j, w, M), \qquad \delta_{\mathcal{C}}(q, x) = (q', w, M), \qquad q' \in Q_j. \tag{26}$$

Every required entry exists. A transition reaches a halt state exactly on an edge to the terminal phase, with motion S.

H.8.3 Annealing Certificate: Arbitrary Binary Inputs

The protocol records finite geometric context. Let $e \in \{0, 2\}$ identify the endpoint row occupied by X. Use $r, j \in \{0, 1, 2\}$ to index tape symbols and $t, a \in \{L, R, S\}$ to denote trits. A vertical search records the last observed trit. Each phase and parameter tuple determines an expert state.

Phase	Expert state	Allowed reading and continuation
Dispatch(e)	0	Read σ_r; jump to selection of r.
Selection jump(e, r)	$1 + r$	Read X; step left to row index $j = e$.
Select(e, r, j)	$1 + r$	Read σ_j; if $j = r$, cross column 1; otherwise change j by $\operatorname{sgn}(r - j)$.
Cross(e, r)	4	Read X if $e = r$, and λ otherwise; enter first source trit.
First source trit(e)	4	Read any trit; enter remaining source-code scan.
Source scan(e)	4	Trit: repeat. Delimiter 0: enter first successor trit.
First successor trit(e)	5	Read a trit t; jump to X for matching t.
Successor scan(e)	5	Trit t: match t. Delimiter 0: start rewind.
Match jump(e, t)	g_t	Read X; step right and begin comparison.
First comparison(e, t)	g_t	Read a. If $a = t$, resume with endpoint e; if $a < t$, search down; if $a > t$, search up.
Search down(t, a)	g_t	Read a or the immediate successor of a. Repeat until t; then resume with endpoint 0.
Search up(t, a)	g_t	Read a or the immediate predecessor of a. Repeat until t; then resume with endpoint 2.
Resume(e, t)	g_t	Read Y; advance to successor scan with endpoint e.
Rewind jump(e)	9	Read X; step left.
Rewind(e)	9	Trit: continue left. Blank: move X here and jump to Y.
Rewind return(e)	9	Read Y; step right to the write field.
Read write(e)	10	Read σ_r; remember this write symbol.
Read motion(e, r)	$11 + r$	Read $M \in \{L, R, S\}$; jump to Z.
Apply(e, r, M)	$h_{r,M}$	Read Z; write σ_r and move M. Dispatch(e) if $M \neq S$; otherwise halt.
Halt	14	No outgoing action.

Here $g_L = 6$, $g_R = 7$, $g_S = 8$, and

$$h_{r,L} = 1 + 2r, \qquad h_{r,R} = 2 + 2r, \qquad h_{r,S} = 11 + r.$$

The protocol starts at Dispatch(0). Searches update the remembered trit after each change. Thus a downward search for S that has seen L must see R before S. The protocol permits unbounded repetition while retaining finite annotations.

Lemma H.29 (Protocol Coverage)**.** *Every expert execution on a correct encoding follows this protocol.*

Proof. Endpoint selection and the following column-1 reading follow from Lemma H.25. Both state codes have positive width. Prefix-block monotonicity and Lemma H.26 give the allowed vertical-search readings and endpoint updates. Rewind reaches the blank in column 1, then Y, the write field, the motion field and Z. These exhaust the phases of one simulated step; induction gives coverage for every finite prefix of the execution. □

The protocol permits additional observation sequences, including unequal lengths for the two code scans. The certificate checks every permitted transition, so this overapproximation preserves the equivalence argument.

Dispatch, Row Selection and Code Scanning For phase i, write Q_i for the controller-state set specified in the following certificate subsections. The simulation relation pairs expert state g_i with controller state $q \in Q_i$, with identical grids, heads, and rune records.

Dispatch. For either endpoint e, phase Dispatch(e) has $Q_i = \{0\}$. On reading $\lambda, 0, 1$, state 0 writes Z, jumps to X, and enters controller state $23, 0, 22$, respectively. These actions match expert state 0.

All Six Row-Selection Cases. The following lists the readings from arrival at X through crossing column 1. The controller-state list gives the state *before each reading*. Each route ends in controller state 10 at the first source-code trit. The table represents endpoint e by the corresponding read symbol, λ or 1.

Endpoint	Wanted	Readings in order	Controller states in order
λ	λ	X, λ, X	$23, 10, 19$
λ	0	$X, \lambda, 0, \lambda$	$0, 20, 7, 25$
λ	1	$X, \lambda, 0, 1, \lambda$	$22, 7, 19, 25, 25$
1	λ	$X, 1, 0, \lambda, \lambda$	$23, 10, 10, 9, 22$
1	0	$X, 1, 0, \lambda$	$0, 20, 4, 25$
1	1	$X, 1, X$	$22, 7, 19$

Each route uses expert state $1 + r$ during selection and state 4 for the final column-1 reading. The X step moves left; comparisons move up or down toward the wanted symbol; the match and crossing steps move right. Every step preserves the read symbol. Substitution in the transition map verifies each route, including successor state 10.

Scan the Source-State Code. The first trit uses controller state 10; subsequent trits use states from $\{4, 18\}$. All of the following writes preserve the read symbol and move right:

Controller state	Read L	Read R	Read S	Read delimiter 0
10	18	4	4	excluded at first trit
4	18	4	4	25
18	18	4	18	25

Entries list successor states. Expert state 4 remains 4 on trits and enters 5 on the delimiter. Thus $Q_i = \{10\}$ at the first source trit, $Q_i = \{4, 18\}$ during the remaining source scan, and $Q_i = \{25\}$ during the successor scan, including the first successor trit.

Successor-Code Dispatch. Controller state 25 has the four relevant transitions

read	L	R	S	0
successor	7	26	15	9

and in every case writes Y and jumps to X. The corresponding expert transitions enter states $6, 7, 8, 9$. The delimiter case occurs only after at least one successor trit. The match jumps therefore use states $7, 26, 15$, and the rewind jump uses state 9, independently of the endpoint.

Trit Matching, Rewind and Tape Updates **Match One Successor Trit.** The match-jump states $7, 26, 15$ preserve X and move right, entering $19, 17, 16$ respectively. The following table gives all comparison cases. Each entry lists $(\text{successor}, \text{written symbol}, \text{motion})$.

Context	State	Read L	Read R	Read S
Target L	19	$(18, X, \mathsf{Y})$	$(19, R, \mathsf{U})$	$(19, S, \mathsf{U})$
Target R	17	$(17, L, \mathsf{D})$	$(2, X, \mathsf{Y})$	$(17, S, \mathsf{U})$
Target S: first comparison	16	$(3, L, \mathsf{D})$	$(5, R, \mathsf{D})$	$(3, X, \mathsf{Y})$
Target S: previous trit L	3	$(3, L, \mathsf{D})$	$(5, R, \mathsf{D})$	excluded
Target S: previous trit R	5	excluded	$(5, R, \mathsf{D})$	$(3, X, \mathsf{Y})$

The corresponding expert states have indices $6, 7, 8$. The exclusions follow from the protocol, especially Lemma H.26. After equality, states $18, 2, 3$ respectively read and preserve Y, then move right to state 25. These states represent Resume(e, L), Resume(e, R), and Resume(e, S). An immediate match retains the endpoint; a downward search selects endpoint 0, and an upward search selects endpoint 2.

Rewind X and Reach the Write Field. Expert state 9 corresponds to the following exact sequence:

Phase	Controller state	Reading	Transition
Rewind jump	9	X	$(14, X, \mathsf{L})$
Rewind through code	14	$t \in \{L, R, S\}$	$(14, t, \mathsf{L})$
Rewind reaches column 1	14	λ	$(19, X, \mathsf{Y})$
Rewind return	19	Y	$(8, Y, \mathsf{R})$

The last step enters expert state 10, so $Q_i = \{8\}$ in phase ReadWrite(e). On reading $\lambda, 0, 1$, state 8 preserves the symbol, moves right and enters $21, 9, 8$ respectively; the expert transitions enter states $11, 12, 13$.

Motion Decoding and Tape Update. For each row below, reading the selected motion writes Y, teleports to Z, and enters the listed controller state. That state then reads Z, writes the indicated symbol and performs exactly the selected move.

Write symbol	Motion-reading state	Apply L	Apply R	Apply S
λ	21	7	4	14
0	9	3	26	18
1	8	11	1	15

The L and R cases return to controller state 0, matching expert state 0. The S cases enter controller halt state 27, matching expert halt state 14. These establish all Apply phases and the terminal pair.

Together, the certificate subsections specify Q_i for every reachable protocol phase and verify every permitted transition. Only source-code scanning requires multivalued sets: $\{4, 18\}$ for each endpoint. All other phase–parameter combinations have a unique controller state.

The certificate contains 91 reachable parameterized phases and 93 related phase–controller pairs, including the terminal pair. For every allowed protocol edge $i \xrightarrow{x} j$ and every controller state $q \in Q_i$, the preceding certificate gives

$$\delta_{\mathcal{E}}(g_i, x) = (g_j, w, M), \qquad \delta_{\mathcal{C}}(q, x) = (q', w, M), \qquad q' \in Q_j. \tag{27}$$

The 162 local checks cover 81 distinct nonterminal entries of the controller table. A transition reaches a halt state exactly on an edge to the terminal phase.

Proof of Theorem H.24. The endpoint and prefix invariant keeps every program search inside the correct code-prefix interval. The no-missing-middle-trit lemma justifies the refined search branches. Both protocols therefore cover every finite expert execution prefix on the respective domains. The metric relation satisfies 160 local identities, and the annealing relation satisfies 162. Equal writes and motions preserve visible cells and hidden rune symbols simultaneously. The prefix form of Lemma H.2 gives equality of complete finite PTOs and of every finite prefix for nonhalting computations. Faithful expert simulation identifies the common execution with the encoded Turing machine. □

Why the Metric Theorem Excludes Empty Input. The metric controller lacks an initial blank transition. Consider a one-state machine with $\delta(0, s) = (0, s, S)$ for every $s \in \Gamma$ and empty tape input. The expert starts with (λ, Z, X), whereas the metric controller cannot take the first step. Hence extending the metric theorem to empty input would give a false claim. Later blank readings remain covered. The annealed controller handles the initial blank and needs no such restriction.

I Short-Input Agreement Does Not Identify an Algorithm

Small training datasets do not force the expert arithmetic rule, even within compact controller classes. We construct two counterexamples using the same alphabets, grid, initial placement, and action vocabulary as the binary experts. Both controllers reproduce every expert PTO within the training-length bounds, including all write and motion decisions. Both nevertheless stop prematurely on longer inputs. Thus the ambiguity persists even with exhaustive supervision over the short-input domain.

Throughout this section, $a, b \in \{0,1\}^+$ denote nonempty binary operand words, with leading zeros permitted. The input $a + b$ or $a\mathtt{x}b$ occupies row zero from column zero; the head starts at the first left-operand digit and every other cell starts blank. Each table entry gives the next state, written symbol, and motion. Motion S writes once and stops; the stopping-state loops specify a total transition map but add no further PTO steps. All state counts include the stopping state.

I.1 A Fourteen-State Spurious Addition Controller

The binary-addition expert uses eleven states, including the stopping state. Replace the initial scanning state by a counter with states $0, 11, 12, 13$, representing zero, one, two, and three scanned left-operand digits. The resulting controller contains fourteen states.

Proposition I.1 (Short-Input Agreement without Exact Addition)**.** *The controller in Table 22 produces the expert PTO on every input $a + b$ with $1 \le |a| \le 3$ and $b \in \{0,1\}^+$, including leading zeros. For every $|a| \ge 4$ and $b \in \{0,1\}^+$, execution stops on the fourth left-operand digit and leaves the input unchanged.*

Proof. Before reaching the separator, the expert remains in state 0 and writes each read digit unchanged while moving right. The counter states produce exactly these writes and motions for the first three digits. On reading $+$, every counter state enters expert state 1 with the same write and left move. No expert transition from states $1, \dots, 10$ returns to state 0. Consequently, the controllers have identical states, grids, and heads after the separator transition and follow the same remaining execution, including the stopping action. If a fourth left-operand digit occurs, state 13 instead writes the digit unchanged and stops, whereas the expert moves right. The unchanged input still contains $+$ and therefore fails the output contract. □

For example, $1111 + 1$ triggers the discrepancy at step four: the expert emits $(1, \mathsf{R})$, while the spurious controller emits $(1, \mathsf{S})$. The expert eventually leaves 10000 on the grid. Thus even all $14^2 = 196$ operand pairs with lengths at most three cannot distinguish these controllers through PTO supervision.

I.2 A Thirty-Eight-State Spurious Multiplication Controller

Let $T_\times$ denote the 32-state binary-multiplication expert. Define $\widetilde{T}_\times$ by retaining all expert transitions and adding six states $32, \dots, 37$ for the initial scan of the left operand. State $32 + j$ records j digits already scanned. The initial state becomes 32; the halt state remains 31. The added states preserve each scanned symbol and move right, except that reading a sixth digit triggers an immediate stop. Reading the separator $\mathtt{x}$ enters expert state 0 with the expert's write and motion.

Proposition I.2. *For every $a \in \{0,1\}^m$ with $1 \le m \le 5$ and every nonempty binary word b, the complete PTOs of $\widetilde{T}_\times$ and $T_\times$ on $a\mathtt{x}b$ agree. For every left operand of at least six digits and every nonempty right operand, the spurious controller stops after six actions and leaves the input expression unchanged.*

Proof. For the first $m \le 5$ digit readings, both controllers preserve the read symbol and move right. The expert remains in state 0, while the spurious controller reaches state $32 + m$. On the separator, both controllers write $\mathtt{x}$, move right, and enter state 0. The grid, head, and control state then coincide, and all remaining transitions agree. For $m \ge 6$, the first five actions agree, but state 37 stops on the sixth digit while the expert moves right. Every preceding action preserves the input, so the final grid still contains the separator and both operands, violating the arithmetic output specification. □

Table 22: **Complete Spurious-Addition Transition Map.** Initial state 0; stopping state 10. Each entry gives $(\text{next state}, \text{write}, \text{motion})$ for the column's read symbol. Motion S writes once and terminates execution.

State	λ	$+$	0	1
0	$(10, \lambda, \mathsf{S})$	$(1, +, \mathsf{L})$	$(11, 0, \mathsf{R})$	$(11, 1, \mathsf{R})$
1	$(8, +, \mathsf{U})$	$(10, +, \mathsf{S})$	$(1, 0, \mathsf{D})$	$(1, 1, \mathsf{D})$
2	$(3, \lambda, \mathsf{L})$	$(2, +, \mathsf{R})$	$(2, 0, \mathsf{R})$	$(2, 1, \mathsf{R})$
3	$(10, \lambda, \mathsf{S})$	$(9, \lambda, \mathsf{D})$	$(4, \lambda, \mathsf{D})$	$(5, \lambda, \mathsf{D})$
4	$(4, \lambda, \mathsf{L})$	$(4, +, \mathsf{U})$	$(7, 0, \mathsf{D})$	$(7, 1, \mathsf{D})$
5	$(5, \lambda, \mathsf{L})$	$(5, +, \mathsf{U})$	$(7, 1, \mathsf{D})$	$(6, 0, \mathsf{L})$
6	$(7, 1, \mathsf{D})$	$(10, +, \mathsf{S})$	$(7, 1, \mathsf{D})$	$(6, 0, \mathsf{L})$
7	$(7, \lambda, \mathsf{R})$	$(1, \lambda, \mathsf{L})$	$(10, 0, \mathsf{S})$	$(10, 1, \mathsf{S})$
8	$(2, 0, \mathsf{R})$	$(10, +, \mathsf{S})$	$(2, 0, \mathsf{R})$	$(2, 1, \mathsf{R})$
9	$(9, \lambda, \mathsf{L})$	$(9, \lambda, \mathsf{U})$	$(10, 0, \mathsf{S})$	$(10, 1, \mathsf{S})$
10	$(10, \lambda, \mathsf{S})$	$(10, +, \mathsf{S})$	$(10, 0, \mathsf{S})$	$(10, 1, \mathsf{S})$
11	$(10, \lambda, \mathsf{S})$	$(1, +, \mathsf{L})$	$(12, 0, \mathsf{R})$	$(12, 1, \mathsf{R})$
12	$(10, \lambda, \mathsf{S})$	$(1, +, \mathsf{L})$	$(13, 0, \mathsf{R})$	$(13, 1, \mathsf{R})$
13	$(10, \lambda, \mathsf{S})$	$(1, +, \mathsf{L})$	$(10, 0, \mathsf{S})$	$(10, 1, \mathsf{S})$

For example, 111111x1 requires the product 111111 (decimal 63), allowing leading-zero padding. The spurious controller instead leaves 111111x1 on the grid and halts at the sixth digit. Exhaustive enumeration of all $62^2 = 3844$ pairs with operand lengths from one through five confirms complete PTO agreement; the argument above additionally covers arbitrary right-operand lengths.

Table 23: Complete Transition Map of the 38-State Spurious Multiplication Controller. Each entry gives $(\text{next state}, \text{written symbol}, \text{motion})$. Initial state 32; halt state 31. Motion S writes once and stops.

State	λ	0	1	x
0	$(1, \lambda, \mathsf{L})$	$(0, 0, \mathsf{R})$	$(0, 1, \mathsf{R})$	$(0, \mathtt{x}, \mathsf{R})$
1	$(31, \lambda, \mathsf{S})$	$(2, 0, \mathsf{L})$	$(2, 1, \mathsf{L})$	$(31, \mathtt{x}, \mathsf{S})$
2	$(31, \lambda, \mathsf{S})$	$(3, 0, \mathsf{D})$	$(4, 1, \mathsf{D})$	$(10, \mathtt{x}, \mathsf{R})$
3	$(6, \lambda, \mathsf{L})$	$(6, 0, \mathsf{L})$	$(6, 1, \mathsf{L})$	$(6, \mathtt{x}, \mathsf{L})$
4	$(7, \lambda, \mathsf{L})$	$(7, 0, \mathsf{L})$	$(7, 1, \mathsf{L})$	$(7, \mathtt{x}, \mathsf{L})$
5	$(31, \lambda, \mathsf{S})$	$(8, 0, \mathsf{L})$	$(8, 1, \mathsf{L})$	$(31, \mathtt{x}, \mathsf{S})$
6	$(9, 0, \mathsf{U})$	$(31, 0, \mathsf{S})$	$(31, 1, \mathsf{S})$	$(31, \mathtt{x}, \mathsf{S})$
7	$(9, 1, \mathsf{U})$	$(31, 0, \mathsf{S})$	$(31, 1, \mathsf{S})$	$(31, \mathtt{x}, \mathsf{S})$
8	$(9, \mathtt{x}, \mathsf{U})$	$(31, 0, \mathsf{S})$	$(31, 1, \mathsf{S})$	$(31, \mathtt{x}, \mathsf{S})$
9	$(0, \lambda, \mathsf{D})$	$(3, 0, \mathsf{D})$	$(4, 1, \mathsf{D})$	$(5, \mathtt{x}, \mathsf{D})$
10	$(31, \lambda, \mathsf{S})$	$(11, \lambda, \mathsf{L})$	$(12, \lambda, \mathsf{L})$	$(31, \mathtt{x}, \mathsf{S})$
11	$(31, \lambda, \mathsf{S})$	$(11, \lambda, \mathsf{L})$	$(11, \lambda, \mathsf{L})$	$(13, \lambda, \mathsf{L})$
12	$(31, \lambda, \mathsf{S})$	$(12, \lambda, \mathsf{L})$	$(12, \lambda, \mathsf{L})$	$(14, \lambda, \mathsf{L})$
13	$(15, 0, \mathsf{R})$	$(13, 0, \mathsf{L})$	$(13, 0, \mathsf{L})$	$(31, \mathtt{x}, \mathsf{S})$
14	$(15, 0, \mathsf{R})$	$(14, 0, \mathsf{L})$	$(14, 1, \mathsf{L})$	$(31, \mathtt{x}, \mathsf{S})$
15	$(16, \lambda, \mathsf{U})$	$(15, 0, \mathsf{R})$	$(15, 1, \mathsf{R})$	$(31, \mathtt{x}, \mathsf{S})$
16	$(18, \lambda, \mathsf{D})$	$(17, 0, \mathsf{R})$	$(17, 1, \mathsf{R})$	$(31, \mathtt{x}, \mathsf{S})$
17	$(10, \lambda, \mathsf{L})$	$(17, 0, \mathsf{R})$	$(17, 1, \mathsf{R})$	$(17, \mathtt{x}, \mathsf{R})$
18	$(19, \lambda, \mathsf{L})$	$(31, 0, \mathsf{S})$	$(31, 1, \mathsf{S})$	$(31, \mathtt{x}, \mathsf{S})$
19	$(20, \lambda, \mathsf{D})$	$(19, 0, \mathsf{L})$	$(19, 1, \mathsf{L})$	$(31, \mathtt{x}, \mathsf{S})$
20	$(30, \lambda, \mathsf{U})$	$(21, 0, \mathsf{R})$	$(21, 1, \mathsf{R})$	$(31, \mathtt{x}, \mathsf{S})$
21	$(28, \lambda, \mathsf{U})$	$(21, 0, \mathsf{R})$	$(21, 1, \mathsf{R})$	$(31, \mathtt{x}, \mathsf{S})$
22	$(19, \lambda, \mathsf{D})$	$(24, \lambda, \mathsf{D})$	$(25, \lambda, \mathsf{D})$	$(31, \mathtt{x}, \mathsf{S})$
23	$(22, \lambda, \mathsf{L})$	$(31, 0, \mathsf{S})$	$(31, 1, \mathsf{S})$	$(31, \mathtt{x}, \mathsf{S})$
24	$(23, 0, \mathsf{U})$	$(23, 0, \mathsf{U})$	$(23, 1, \mathsf{U})$	$(31, \mathtt{x}, \mathsf{S})$
25	$(23, 1, \mathsf{U})$	$(23, 1, \mathsf{U})$	$(27, 0, \mathsf{U})$	$(31, \mathtt{x}, \mathsf{S})$

Continued on the next page.

State	λ	0	1	x
26	$(25, \lambda, \mathsf{D})$	$(25, \lambda, \mathsf{D})$	$(29, \lambda, \mathsf{D})$	$(31, \mathtt{x}, \mathsf{S})$
27	$(26, \lambda, \mathsf{L})$	$(31, 0, \mathsf{S})$	$(31, 1, \mathsf{S})$	$(31, \mathtt{x}, \mathsf{S})$
28	$(22, \lambda, \mathsf{L})$	$(28, 0, \mathsf{R})$	$(28, 1, \mathsf{R})$	$(31, \mathtt{x}, \mathsf{S})$
29	$(27, 0, \mathsf{U})$	$(27, 0, \mathsf{U})$	$(27, 1, \mathsf{U})$	$(31, \mathtt{x}, \mathsf{S})$
30	$(31, \lambda, \mathsf{S})$	$(31, 0, \mathsf{S})$	$(31, 1, \mathsf{S})$	$(31, \mathtt{x}, \mathsf{S})$
31	$(31, \lambda, \mathsf{S})$	$(31, 0, \mathsf{S})$	$(31, 1, \mathsf{S})$	$(31, \mathtt{x}, \mathsf{S})$
32	$(1, \lambda, \mathsf{L})$	$(33, 0, \mathsf{R})$	$(33, 1, \mathsf{R})$	$(0, \mathtt{x}, \mathsf{R})$
33	$(1, \lambda, \mathsf{L})$	$(34, 0, \mathsf{R})$	$(34, 1, \mathsf{R})$	$(0, \mathtt{x}, \mathsf{R})$
34	$(1, \lambda, \mathsf{L})$	$(35, 0, \mathsf{R})$	$(35, 1, \mathsf{R})$	$(0, \mathtt{x}, \mathsf{R})$
35	$(1, \lambda, \mathsf{L})$	$(36, 0, \mathsf{R})$	$(36, 1, \mathsf{R})$	$(0, \mathtt{x}, \mathsf{R})$
36	$(1, \lambda, \mathsf{L})$	$(37, 0, \mathsf{R})$	$(37, 1, \mathsf{R})$	$(0, \mathtt{x}, \mathsf{R})$
37	$(1, \lambda, \mathsf{L})$	$(31, 0, \mathsf{S})$	$(31, 1, \mathsf{S})$	$(0, \mathtt{x}, \mathsf{R})$

I.3 Capacity Does Not Exclude the Counterexamples

The constructions add only three states to the eleven-state addition expert and six states to the thirty-two-state multiplication expert. Every training subset within the stated length bounds supplies identical PTO targets for the expert and the corresponding spurious controller.

Theorem 2.5 represents the two controllers using ordinary DFST dimensions 14 and 38. For Softmax DFSTs with finite logits in exact real arithmetic, dimensions 15 and 39 suffice for exact argmax behavior. These bounds provide sufficient capacities, without asserting minimality. Padding the discrete controllers with unreachable states extends the construction to every larger dimension. In particular, the addition sweeps at dimensions 22 and 33 and the multiplication sweeps at dimensions 64 and 96 can express the counterexamples.

Action agreement and fitting the one-hot training targets require separate statements. The exact-real softmax construction in Section B guarantees the correct argmax actions, but need not produce one-hot readouts. For any PTO dataset within the stated training-length bounds, softmax logits approaching the deterministic vertices instead drive the MSE arbitrarily close to zero: finite-horizon recurrence and readout maps depend continuously on the effective parameters. Under rounded arithmetic that saturates softmax at vertices and preserves vertex operations, the one-hot realization gives exactly zero training MSE. One explicit choice assigns logit 0 to each selected transition, output, and initial-state coordinate, and -128 to every other coordinate under correctly rounded binary32 exponentiation.

Table 24: **Generalization despite Capacity for Short-Input Counterexamples.** The wider Softmax DFST sweeps each contain six configurations. Five reach the robust testing cap in each task.

Task	Expert states	Spurious states	Tested dimensions	Cap-reaching runs
`add2`	11	14	$22, 33$	$5/6$
`mult2`	32	38	$64, 96$	$5/6$

The wider sweeps exclude insufficient capacity as the explanation for avoiding these two spurious rules (Table 24 and fig. 9). The experiments support a useful implicit bias of parametrization, initialization, and gradient-based optimization toward length-independent behavior. However, no claim identifies the failed runs with the constructed counterexamples, and the results do not establish that gradient descent always selects the simplest controller. Extracted state counts alone need not capture the relevant notion of simplicity.

I.4 A Simple Model for Studying Algorithmic Learning

The ordinary DFST uses three input-indexed tensors and an initial-state vector. Each recurrence and readout consists of a matrix-vector product, without nonlinear activations. This stepwise multilinearity offers a transparent starting point for analyzing which solutions gradient-based optimization favors. Repeated use of shared transition parameters across a trajectory still produces a nonconvex learning

problem; neither the complete trajectory map nor the MSE objective generally remains multilinear in the shared parameters. The softmax parametrization adds another nonlinearity through the logits.

NeurIPS Paper Checklist

1. **Claims**

   Question: Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope?

   Answer: [Yes]

   Justification: In the abstract and introduction, we claim that we introduce a simple learning framework capable of achieving extreme length generalization and exact learning on arithmetic and universal simulation tasks. We substantiate these claims with experimental results and mathematical proofs.

   Guidelines:

   - The answer [N/A] means that the abstract and introduction do not include the claims made in the paper.
   - The abstract and/or introduction should clearly state the claims made, including the contributions made in the paper and important assumptions and limitations. A [No] or [N/A] answer to this question will not be perceived well by the reviewers.
   - The claims made should match theoretical and experimental results, and reflect how much the results can be expected to generalize to other settings.
   - It is fine to include aspirational goals as motivation as long as it is clear that these goals are not attained by the paper.

2. **Limitations**

   Question: Does the paper discuss the limitations of the work performed by the authors?

   Answer: [Yes]

   Justification: We provide evidence of hyperparameter configurations where our differentiable controllers cannot achieve extreme length generalization and provably do not exactly learn the ground-truth procedure.

   Guidelines:

   - The answer [N/A] means that the paper has no limitation while the answer [No] means that the paper has limitations, but those are not discussed in the paper.
   - The authors are encouraged to create a separate "Limitations" section in their paper.
   - The paper should point out any strong assumptions and how robust the results are to violations of these assumptions (e.g., independence assumptions, noiseless settings, model well-specification, asymptotic approximations only holding locally). The authors should reflect on how these assumptions might be violated in practice and what the implications would be.
   - The authors should reflect on the scope of the claims made, e.g., if the approach was only tested on a few datasets or with a few runs. In general, empirical results often depend on implicit assumptions, which should be articulated.
   - The authors should reflect on the factors that influence the performance of the approach. For example, a facial recognition algorithm may perform poorly when image resolution is low or images are taken in low lighting. Or a speech-to-text system might not be used reliably to provide closed captions for online lectures because it fails to handle technical jargon.
   - The authors should discuss the computational efficiency of the proposed algorithms and how they scale with dataset size.
   - If applicable, the authors should discuss possible limitations of their approach to address problems of privacy and fairness.
   - While the authors might fear that complete honesty about limitations might be used by reviewers as grounds for rejection, a worse outcome might be that reviewers discover limitations that aren't acknowledged in the paper. The authors should use their best judgment and recognize that individual actions in favor of transparency play an important role in developing norms that preserve the integrity of the community. Reviewers will be specifically instructed to not penalize honesty concerning limitations.

3. **Theory assumptions and proofs**

Question: For each theoretical result, does the paper provide the full set of assumptions and a complete (and correct) proof?

Answer: [Yes]

Justification: We provide the full set of empirically falsifiable assumptions for our theorems. All theorems are carefully proved in the appendix sections.

Guidelines:

- The answer [N/A] means that the paper does not include theoretical results.
- All the theorems, formulas, and proofs in the paper should be numbered and cross-referenced.
- All assumptions should be clearly stated or referenced in the statement of any theorems.
- The proofs can either appear in the main paper or the supplemental material, but if they appear in the supplemental material, the authors are encouraged to provide a short proof sketch to provide intuition.
- Inversely, any informal proof provided in the core of the paper should be complemented by formal proofs provided in appendix or supplemental material.
- Theorems and Lemmas that the proof relies upon should be properly referenced.

4. **Experimental result reproducibility**

Question: Does the paper fully disclose all the information needed to reproduce the main experimental results of the paper to the extent that it affects the main claims and/or conclusions of the paper (regardless of whether the code and data are provided or not)?

Answer: [Yes]

Justification: In our appendix section about implementation details, we provide full details on our training procedure. Furthermore, we make our code publicly available (alongside the weights of the runs discussed in the paper) so that other researchers can experiment with and visualize learning from Turing-machine computation.

Guidelines:

- The answer [N/A] means that the paper does not include experiments.
- If the paper includes experiments, a [No] answer to this question will not be perceived well by the reviewers: Making the paper reproducible is important, regardless of whether the code and data are provided or not.
- If the contribution is a dataset and/or model, the authors should describe the steps taken to make their results reproducible or verifiable.
- Depending on the contribution, reproducibility can be accomplished in various ways. For example, if the contribution is a novel architecture, describing the architecture fully might suffice, or if the contribution is a specific model and empirical evaluation, it may be necessary to either make it possible for others to replicate the model with the same dataset, or provide access to the model. In general. releasing code and data is often one good way to accomplish this, but reproducibility can also be provided via detailed instructions for how to replicate the results, access to a hosted model (e.g., in the case of a large language model), releasing of a model checkpoint, or other means that are appropriate to the research performed.
- While NeurIPS does not require releasing code, the conference does require all submissions to provide some reasonable avenue for reproducibility, which may depend on the nature of the contribution. For example
  (a) If the contribution is primarily a new algorithm, the paper should make it clear how to reproduce that algorithm.
  (b) If the contribution is primarily a new model architecture, the paper should describe the architecture clearly and fully.
  (c) If the contribution is a new model (e.g., a large language model), then there should either be a way to access this model for reproducing the results or a way to reproduce the model (e.g., with an open-source dataset or instructions for how to construct the dataset).

(d) We recognize that reproducibility may be tricky in some cases, in which case authors are welcome to describe the particular way they provide for reproducibility. In the case of closed-source models, it may be that access to the model is limited in some way (e.g., to registered users), but it should be possible for other researchers to have some path to reproducing or verifying the results.

5. **Open access to data and code**

   Question: Does the paper provide open access to the data and code, with sufficient instructions to faithfully reproduce the main experimental results, as described in supplemental material?

   Answer: [Yes]

   Justification: We provide a well documented codebase with sufficient instructions.

   Guidelines:

   - The answer [N/A] means that paper does not include experiments requiring code.
   - Please see the NeurIPS code and data submission guidelines (`https://neurips.cc/public/guides/CodeSubmissionPolicy`) for more details.
   - While we encourage the release of code and data, we understand that this might not be possible, so [No] is an acceptable answer. Papers cannot be rejected simply for not including code, unless this is central to the contribution (e.g., for a new open-source benchmark).
   - The instructions should contain the exact command and environment needed to run to reproduce the results. See the NeurIPS code and data submission guidelines (`https://neurips.cc/public/guides/CodeSubmissionPolicy`) for more details.
   - The authors should provide instructions on data access and preparation, including how to access the raw data, preprocessed data, intermediate data, and generated data, etc.
   - The authors should provide scripts to reproduce all experimental results for the new proposed method and baselines. If only a subset of experiments are reproducible, they should state which ones are omitted from the script and why.
   - At submission time, to preserve anonymity, the authors should release anonymized versions (if applicable).
   - Providing as much information as possible in supplemental material (appended to the paper) is recommended, but including URLs to data and code is permitted.

6. **Experimental setting/details**

   Question: Does the paper specify all the training and test details (e.g., data splits, hyperparameters, how they were chosen, type of optimizer) necessary to understand the results?

   Answer: [Yes]

   Justification: We completely specify all details in the appendix section about implementations. We also provide the code, hyperparameter configurations, and trained final weights in our GitHub repository.

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.
   - The experimental setting should be presented in the core of the paper to a level of detail that is necessary to appreciate the results and make sense of them.
   - The full details can be provided either with the code, in appendix, or as supplemental material.

7. **Experiment statistical significance**

   Question: Does the paper report error bars suitably and correctly defined or other appropriate information about the statistical significance of the experiments?

   Answer: [Yes]

   Justification: Our DSFT models are deterministically initialized so the only source of randomness comes from the dataset generation. We achieve our length-generalization results across all tasks on many different random seeds (details in the appendix).

   Guidelines:

- The answer [N/A] means that the paper does not include experiments.
- The authors should answer [Yes] if the results are accompanied by error bars, confidence intervals, or statistical significance tests, at least for the experiments that support the main claims of the paper.
- The factors of variability that the error bars are capturing should be clearly stated (for example, train/test split, initialization, random drawing of some parameter, or overall run with given experimental conditions).
- The method for calculating the error bars should be explained (closed form formula, call to a library function, bootstrap, etc.)
- The assumptions made should be given (e.g., Normally distributed errors).
- It should be clear whether the error bar is the standard deviation or the standard error of the mean.
- It is OK to report 1-sigma error bars, but one should state it. The authors should preferably report a 2-sigma error bar than state that they have a 96% CI, if the hypothesis of Normality of errors is not verified.
- For asymmetric distributions, the authors should be careful not to show in tables or figures symmetric error bars that would yield results that are out of range (e.g., negative error rates).
- If error bars are reported in tables or plots, the authors should explain in the text how they were calculated and reference the corresponding figures or tables in the text.

8. **Experiments compute resources**

   Question: For each experiment, does the paper provide sufficient information on the computer resources (type of compute workers, memory, time of execution) needed to reproduce the experiments?

   Answer: [Yes]

   Justification: As we mention in the appendix, we trained on single NVIDIA A100-SXM4-40GB and Tesla V100-SXM2-32GB GPUs.

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.
   - The paper should indicate the type of compute workers CPU or GPU, internal cluster, or cloud provider, including relevant memory and storage.
   - The paper should provide the amount of compute required for each of the individual experimental runs as well as estimate the total compute.
   - The paper should disclose whether the full research project required more compute than the experiments reported in the paper (e.g., preliminary or failed experiments that didn't make it into the paper).

9. **Code of ethics**

   Question: Does the research conducted in the paper conform, in every respect, with the NeurIPS Code of Ethics `https://neurips.cc/public/EthicsGuidelines`?

   Answer: [Yes]

   Justification: We carefully reviewed the ethics guidelines and confirm that our research respects them.

   Guidelines:

   - The answer [N/A] means that the authors have not reviewed the NeurIPS Code of Ethics.
   - If the authors answer [No], they should explain the special circumstances that require a deviation from the Code of Ethics.
   - The authors should make sure to preserve anonymity (e.g., if there is a special consideration due to laws or regulations in their jurisdiction).

10. **Broader impacts**

    Question: Does the paper discuss both potential positive societal impacts and negative societal impacts of the work performed?

Answer: [N/A]

Justification: This paper presents work whose goal is to advance the field of machine learning. There are many potential societal consequences of our work, none of which we feel must be specifically highlighted here.

Guidelines:

- The answer [N/A] means that there is no societal impact of the work performed.
- If the authors answer [N/A] or [No], they should explain why their work has no societal impact or why the paper does not address societal impact.
- Examples of negative societal impacts include potential malicious or unintended uses (e.g., disinformation, generating fake profiles, surveillance), fairness considerations (e.g., deployment of technologies that could make decisions that unfairly impact specific groups), privacy considerations, and security considerations.
- The conference expects that many papers will be foundational research and not tied to particular applications, let alone deployments. However, if there is a direct path to any negative applications, the authors should point it out. For example, it is legitimate to point out that an improvement in the quality of generative models could be used to generate Deepfakes for disinformation. On the other hand, it is not needed to point out that a generic algorithm for optimizing neural networks could enable people to train models that generate Deepfakes faster.
- The authors should consider possible harms that could arise when the technology is being used as intended and functioning correctly, harms that could arise when the technology is being used as intended but gives incorrect results, and harms following from (intentional or unintentional) misuse of the technology.
- If there are negative societal impacts, the authors could also discuss possible mitigation strategies (e.g., gated release of models, providing defenses in addition to attacks, mechanisms for monitoring misuse, mechanisms to monitor how a system learns from feedback over time, improving the efficiency and accessibility of ML).

11. **Safeguards**

Question: Does the paper describe safeguards that have been put in place for responsible release of data or models that have a high risk for misuse (e.g., pre-trained language models, image generators, or scraped datasets)?

Answer: [N/A]

Justification: We don't believe our research holds such risks.

Guidelines:

- The answer [N/A] means that the paper poses no such risks.
- Released models that have a high risk for misuse or dual-use should be released with necessary safeguards to allow for controlled use of the model, for example by requiring that users adhere to usage guidelines or restrictions to access the model or implementing safety filters.
- Datasets that have been scraped from the Internet could pose safety risks. The authors should describe how they avoided releasing unsafe images.
- We recognize that providing effective safeguards is challenging, and many papers do not require this, but we encourage authors to take this into account and make a best faith effort.

12. **Licenses for existing assets**

Question: Are the creators or original owners of assets (e.g., code, data, models), used in the paper, properly credited and are the license and terms of use explicitly mentioned and properly respected?

Answer: [Yes]

Justification: We, the authors, are the original owners of the assets used in the paper.

Guidelines:

- The answer [N/A] means that the paper does not use existing assets.

- The authors should cite the original paper that produced the code package or dataset.
- The authors should state which version of the asset is used and, if possible, include a URL.
- The name of the license (e.g., CC-BY 4.0) should be included for each asset.
- For scraped data from a particular source (e.g., website), the copyright and terms of service of that source should be provided.
- If assets are released, the license, copyright information, and terms of use in the package should be provided. For popular datasets, `paperswithcode.com/datasets` has curated licenses for some datasets. Their licensing guide can help determine the license of a dataset.
- For existing datasets that are re-packaged, both the original license and the license of the derived asset (if it has changed) should be provided.
- If this information is not available online, the authors are encouraged to reach out to the asset's creators.

13. **New assets**

    Question: Are new assets introduced in the paper well documented and is the documentation provided alongside the assets?

    Answer: [Yes]

    Justification: We believe that the codebase for our paper is well documented.

    Guidelines:

    - The answer [N/A] means that the paper does not release new assets.
    - Researchers should communicate the details of the dataset/code/model as part of their submissions via structured templates. This includes details about training, license, limitations, etc.
    - The paper should discuss whether and how consent was obtained from people whose asset is used.
    - At submission time, remember to anonymize your assets (if applicable). You can either create an anonymized URL or include an anonymized zip file.

14. **Crowdsourcing and research with human subjects**

    Question: For crowdsourcing experiments and research with human subjects, does the paper include the full text of instructions given to participants and screenshots, if applicable, as well as details about compensation (if any)?

    Answer: [N/A]

    Justification: The paper does not involve crowdsourcing nor research with human subjects.

    Guidelines:

    - The answer [N/A] means that the paper does not involve crowdsourcing nor research with human subjects.
    - Including this information in the supplemental material is fine, but if the main contribution of the paper involves human subjects, then as much detail as possible should be included in the main paper.
    - According to the NeurIPS Code of Ethics, workers involved in data collection, curation, or other labor should be paid at least the minimum wage in the country of the data collector.

15. **Institutional review board (IRB) approvals or equivalent for research with human subjects**

    Question: Does the paper describe potential risks incurred by study participants, whether such risks were disclosed to the subjects, and whether Institutional Review Board (IRB) approvals (or an equivalent approval/review based on the requirements of your country or institution) were obtained?

    Answer: [N/A]

    Justification: The paper does not involve crowdsourcing nor research with human subjects.

Guidelines:

- The answer [N/A] means that the paper does not involve crowdsourcing nor research with human subjects.
- Depending on the country in which research is conducted, IRB approval (or equivalent) may be required for any human subjects research. If you obtained IRB approval, you should clearly state this in the paper.
- We recognize that the procedures for this may vary significantly between institutions and locations, and we expect authors to adhere to the NeurIPS Code of Ethics and the guidelines for their institution.
- For initial submissions, do not include any information that would break anonymity (if applicable), such as the institution conducting the review.

16. **Declaration of LLM usage**

    Question: Does the paper describe the usage of LLMs if it is an important, original, or non-standard component of the core methods in this research? Note that if the LLM is used only for writing, editing, or formatting purposes and does *not* impact the core methodology, scientific rigor, or originality of the research, declaration is not required.

    Answer: [Yes]

    Justification: We used LLMs for code writing and for the generated controller equivalence proofs. The proofs have been carefully reviewed by the authors of the paper.

    Guidelines:

    - The answer [N/A] means that the core method development in this research does not involve LLMs as any important, original, or non-standard components.
    - Please refer to our LLM policy in the NeurIPS handbook for what should or should not be described.