Learning Bimachines from Aligned Examples

Mans Hulden MHULDEN@NCF.EDU
New College of Florida
5800 Bay Shore Rd.
Sarasota, FL 34243, USA

Michael Ginn MICHAEL.GINN@COLORADO.EDU
University of Colorado
430 UCB
Boulder, CO 80309, USA

Abstract

Bimachines provide a deterministic representation of functional string transductions by combining a left-to-right DFA, a right-to-left DFA, and a local output function depending on both contexts. We study an aligned learning regime in which each training example provides one output chunk per input position.

In this setting we present a practical bimachine learner based on RPNI-style state merging: it starts from left and right prefix-tree automata, constructs a partial local output table from aligned chunks, alternates merge-and-fold passes on the two sides, validates merges against training constraints and optional witness queries, and finally completes and minimizes the resulting hypothesis as far as the available evidence permits.

The learner supports deletions and anchored insertions without input-ϵ moves. Experiments on a battery of local, global, and length-changing synthetic transductions recover compact bimachines with perfect chunk-level agreement on 500 random test strings per task in an aligned-oracle setting. In a separate sample-only regime on randomly generated target bimachines, held-out agreement rises with more aligned examples and falls with larger target machines, while exact structural recovery emerges first on the smallest families.

Keywords: Grammatical inference, bimachines, finite-state transductions, state merging, aligned learning

1. Introduction

Finite-state transducers are widely used in rewriting (Kaplan and Kay, 1994), language technology (Mohri, 1997), and symbolic processing. For *functional* transductions, bimachines (Schützenberger, 1961) are especially attractive because they separate left context, right context, and local output behavior. This yields interpretable deterministic models, but it also raises a learning problem: how can one recover both automata and a local output table from examples?

This paper focuses on an aligned regime, where each training example comes with one output chunk per input position.[1] Under this assumption, local consistency constraints can be imposed directly on the bimachine output table. That makes state-merging learning feasible: one can start from prefix-tree automata on the left and right, merge states subject to deterministic fold closure and output consistency, and then minimize the learned structure.

1. The learner, experiment scripts, and an interactive browser demo of the algorithm are available at https://github.com/mhulden/bimachine-learner/.

© 2026 M. Hulden & M. Ginn.

The closest learning traditions are state-merging algorithms for subsequential transducers such as OSTIA (Onward Subsequential Transducer Inference Algorithm; Oncina et al., 1993) and theoretical algorithms for learning rational functions via deterministic transducers with lookahead (Boiret et al., 2012). Bimachines have also been used as deterministic compilation targets for rule systems (Skut et al., 2004). To our knowledge, however, direct state-merging learning of the two context automata and local output table of a bimachine from aligned examples has not previously been developed.

The aligned setting is not purely artificial. It is appropriate whenever the teacher provides an explicit alignment between input positions and output chunks, for example in hand-labeled rewrite data, in symbolic transformation systems whose local effect is observable, or in pipelines where an external model supplies an alignment. Learning such alignments from unaligned input-output pairs is an important future direction. In this paper we assume the alignments are always given. We establish empirical performance in this aligned setting; we do not claim identification in the limit or globally minimal hypotheses for the present greedy learner.

Contributions.

- We formulate aligned bimachine learning with anchored chunks $y_i \in \Gamma^*$, allowing deletions and anchored insertions without input-ϵ moves.
- We present a practical learner in the style of RPNI (Regular Positive and Negative Inference; Oncina and García, 1992) that alternates merge-and-fold passes on the left and right automata, rather than reducing the problem to a one-sided transducer learner, and validates candidate merges against the aligned data and, in the oracle-assisted mode, against witness queries.
- We describe output-table completion and context-automaton minimization, which together recover compact bimachines on global and two-sided tasks.
- We experimentally validate the learner in two aligned regimes: an oracle-assisted benchmark battery with perfect chunk-level agreement on 500 random test strings for each of ten synthetic tasks, and a sample-only random-target study that quantifies the tradeoff between data, target size, and exact structural recovery.

2. Background and notation

Let Σ be a finite input alphabet and Γ a finite output alphabet. We write Σ^* and Γ^* for the free monoids over these alphabets. An *aligned transduction* assigns to each input string $x = x_1 \cdots x_n \in \Sigma^*$ a chunk sequence

$$O(x) = (y_1, \ldots, y_n) \in (\Gamma^*)^n.$$

Its underlying string transduction is obtained by concatenating the chunks:

$$f_O(x) = y_1 \cdots y_n.$$

Thus an aligned sample is a pair $(x, O(x))$. Deletions are represented by $y_i = \epsilon$, and anchored insertions are represented by chunks with $|y_i| > 1$.

2.1. Bimachines

A deterministic bimachine is a triple

$$\mathcal{B} = (A_L, A_R, \omega),$$

where $A_L = (Q_L, \Sigma, \delta_L, q_L^0)$ is a left-to-right DFA, $A_R = (Q_R, \Sigma, \delta_R, p_R^0)$ is a right-to-left DFA, and

$$\omega : Q_L \times \Sigma \times Q_R \to \Gamma^*$$

is a local output function. Given $x = x_1 \cdots x_n$, let

$$q_i = \delta_L(q_L^0, x_1 \cdots x_{i-1}) \quad \text{and} \quad p_i = \delta_R(p_R^0, x_n \cdots x_{i+1}).$$

The chunk emitted at position i is

$$\omega(q_i, x_i, p_i),$$

and the final output is the concatenation of these chunks. For simplicity we use a per-symbol form without boundary outputs (no separate initial or final output); all benchmarks in this paper satisfy $f_O(\epsilon) = \epsilon$. During learning, and in hypotheses returned by the sample-only learner, δ_L, δ_R, and ω may be partial; oracle-assisted post-processing makes them total (Section 3.5). Thus the aligned learning problem is exactly the problem of recovering both automata and the table ω. See Figure 1 for an illustration.

The classical rational-function characterization uses general bimachines with initial and final boundary outputs. Our per-symbol form is a restriction of that setting. Classical minimization results for rational word functions motivate learning bimachines directly rather than through a more opaque transducer normal form (Reutenauer and Schützenberger, 1991; Roche, 1997). For the present paper, the key structural point is that all output behavior is localized in ω once left and right contexts have been represented finitely.

3. Learning from aligned chunks

We assume access to a finite aligned training set

$$D = \{(x, O(x))\},$$

where $O(x)$ is the chunk sequence defined above. Optionally, a *chunk oracle* may be queried on fresh strings and returns $O(x)$ directly. This is stronger than a translation oracle, which would return only the concatenated output $f_O(x)$. The same implementation also supports a sample-only variant that omits oracle queries during learning and relies only on the observed aligned examples; we use that mode in the random-target experiments of Section 4.4.

3.1. Overview

The learner is RPNI-style state merging (Oncina and García, 1992) adapted to the two-sided bimachine setting; de la Higuera (2010) gives a textbook presentation of RPNI and OSTIA. Recall that a red/blue state-merging learner for automata (Lang et al., 1998) proceeds as follows:

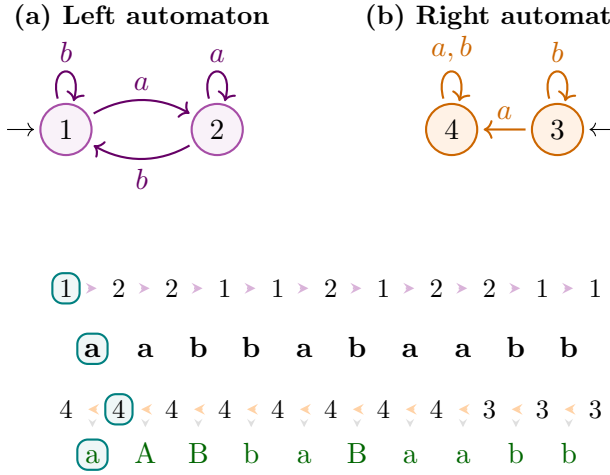


Figure 1: Illustration of a bimachine and its trace on the input **aabbabaabb**. The objective is to capitalize the input letters **a** and **b** only if two conditions are met simultaneously: their immediate left neighbor is an **a**, and another **a** appears somewhere later in the remaining string. Left state 2 records that the previous symbol was **a**; right state 4 records that an **a** occurs further right. An entry such as $(1, a, 3/4)$ stands for both right states.

- Build an initial prefix-tree automaton that exactly represents the observed sample.
- Mark the initial state red and maintain a blue frontier of states reachable in one transition from red states.
- Try to merge a blue state into an existing red state, accepting the merge only if the resulting hypothesis remains consistent with the available evidence.
- If no red target gives an admissible merge, promote the blue state to red.
- Repeat until the frontier is exhausted or no further admissible merges are found.

States are ordered by their *representatives*, the shortlex-least strings reaching them (first by length, then lexicographically), which fixes a deterministic exploration order.

Our bimachine learner applies this idea on both sides: it builds a left prefix tree over input prefixes and a right prefix tree over reversed inputs, initializes a partial output table from the aligned chunks, and alternates left and right merge passes.

Algorithm 1 summarizes the overall pipeline; we now unpack its main stages. In sample-only mode the oracle O is absent: the merge passes skip witness validation, and CompleteOmega fills entries only from representative strings that occur in the training sample.

Greedy restarts. Within each left or right pass, the learner follows the usual greedy restart policy from state-merging algorithms: as soon as one candidate merge is accepted, the pass restarts from the updated hypothesis and recomputes the red/blue frontier. This

Algorithm 1: LearnBimachine$(\Sigma, D, O; k, L_{\text{cont}}, L_{\text{head}}, maxRounds)$
Input: Alphabet Σ; aligned samples $D = \{(x, O(x))\}$; chunk oracle O (optional)
Output: Bimachine $\mathcal{B} = (A_L, A_R, \omega)$

$A_L \leftarrow$ BuildLeftPTA(Σ, D)
$A_R \leftarrow$ BuildRightPTA(Σ, D)
$\omega \leftarrow$ InitOmegaFromData(A_L, A_R, D)
for $t \leftarrow 1$ **to** $maxRounds$ **do**
    changed $\leftarrow$ **false**
    $(A_L, A_R, \omega, \Delta) \leftarrow$ MergePassLeft$(A_L, A_R, \omega, \Sigma, D, O; k, L_{\text{cont}})$
    changed $\leftarrow$ changed $\vee\Delta$
    $(A_L, A_R, \omega, \Delta) \leftarrow$ MergePassRight$(A_L, A_R, \omega, \Sigma, D, O; k, L_{\text{head}})$
    changed $\leftarrow$ changed $\vee\Delta$
    if not changed **then**
        break
$(A_L, A_R, \omega) \leftarrow$ AlternateMinimize$(\Sigma, A_L, A_R, \omega, O)$
return (A_L, A_R, ω)

behavior is implemented explicitly and matters in practice because accepted merges often expose new fold closures and new merge opportunities immediately.

3.2. Prefix-tree initialization

The left prefix-tree automaton contains one state per distinct training prefix. The right prefix-tree automaton is built on reversed strings and therefore contains one state per distinct training suffix. From an aligned sample $(x, (y_1, \ldots, y_n))$ we record, for each position i,

$$\omega(q_i, x_i, p_i) = y_i.$$

If the same triple (q_i, x_i, p_i) is forced to take two different values, the aligned data is inconsistent with any functional bimachine.

3.3. Merge-and-fold

A source state and target state are selected as a candidate for merging from the sets of blue and red states, respectively. Because the automata are deterministic, this may introduce transition conflicts: two outgoing transitions on the same symbol may now point to different states. We resolve these by classical fold closure, recursively merging conflicting successors. A merge is rejected as soon as fold closure creates an inconsistency in the partial output table. This is the same structural idea used in RPNI for languages, but here it must coexist with the local output constraints induced by aligned chunks.

3.4. Witness validation

Training examples alone are rarely sufficient to block all harmful merges, so we also validate against sets of *witness strings*. For a left-merge candidate $u \to r$, let $\pi(u)$ be the representative of u in the current left automaton; the witness set $W_L(u)$ consists of strings $\pi(u)z$ with

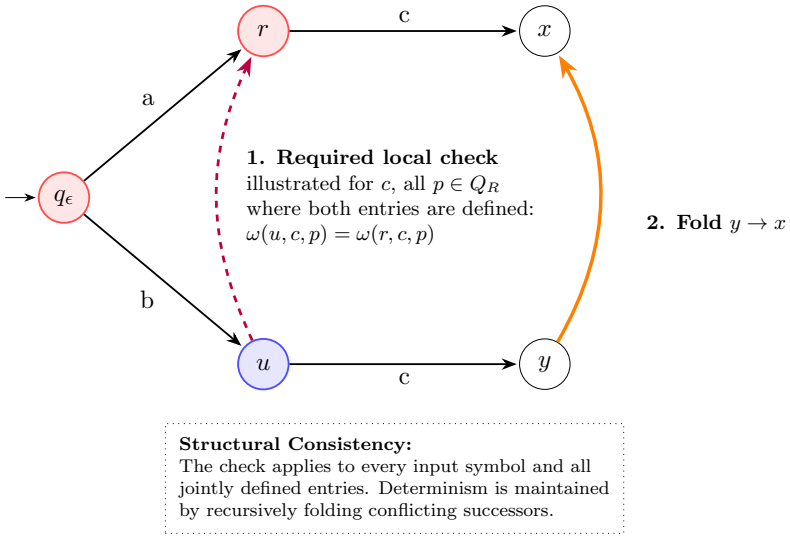


Figure 2: Illustration of merge-and-fold in the left DFA A_L. A blue state u is speculatively merged into a red state r; the displayed output check is only for symbol c. Acceptance requires consistency for every symbol and jointly defined entry, all identifications forced by fold closure, the training sample, and witness queries. Here determinism forces the c-successor y to be folded into x.

continuations $1 \leq |z| \leq L_{\text{cont}}$ that are not already in the training sample, at most k of them. Symmetrically, for a right-merge candidate with representative suffix $\sigma(u)$, $W_R(u)$ consists of strings $h\sigma(u)$ with heads $1 \leq |h| \leq L_{\text{head}}$. The oracle-assisted experiments use $k = 80$ and $L_{\text{cont}} = L_{\text{head}} = 3$; for the alphabets used here ($|\Sigma| \leq 4$) this budget covers all, or all but a few, such continuations and heads, so the witness sets are essentially exhaustive. A merge is accepted only if the resulting hypothesis remains consistent with both the training set and all witness queries. In practice these witnesses are essential for global begin/end conditions and other two-sided tasks.

Right pass. MergePassRight is symmetric, operating on A_R and using witness set $W_R(u)$. The important design choice is that the learner alternates left and right passes rather than saturating one side completely before touching the other. Empirically this keeps the two context structures synchronized, which matters on tasks where left merges expose new right-side redundancies and vice versa.

Validation. ValidateAndAugment checks both sources of evidence available in the aligned regime. First, all training samples in D must remain consistent with the current partial table ω. Second, the hypothesis must agree with the chunk oracle on the witness set associated with the proposed merge. When a merge is accepted, paths for the queried strings and their chunks are added to the current automata and output table as additional constraints. Because earlier merges may already have identified states, these augmented automata are no longer necessarily prefix trees. We use strict validation: a candidate merge is rejected as soon as it disagrees on any anchored chunk.

Algorithm 2: MergePassLeft$(A_L, A_R, \omega, \Sigma, D, O; k, L_{\text{cont}})$ (red/blue order)
Output: Updated $(A_L, A_R, \omega, \Delta)$

$\Delta \leftarrow$ **false**
repeat
  restart $\leftarrow$ **false**
  Initialize red set $R \leftarrow \{q_L^0\}$
  while the red/blue frontier is nonempty **do**
    Pick blue u by shortlex order of its representative $\pi(u)$
    for each $r \in R$ in shortlex order **do**
      $(A'_L, A'_R, \omega') \leftarrow$ TryMergeLeft(A_L, A_R, ω, r, u)
      if $(A'_L, A'_R, \omega') \neq \perp$ **and**
        ValidateAndAugment$(A'_L, A'_R, \omega', D, O, W_L(u))$ **then**
        $(A_L, A_R, \omega) \leftarrow (A'_L, A'_R, \omega')$ with witness paths added
        $\Delta \leftarrow$ **true**; restart $\leftarrow$ **true**
        break
    if restart **then break**
    Promote u to red: $R \leftarrow R \cup \{u\}$
until not restart
return $(A_L, A_R, \omega, \Delta)$

3.5. Output-table completion and minimization

The merge-and-fold phase usually yields a compact and accurate bimachine, but it can still contain redundant distinctions. Some triples (q, a, p) may never have been exercised by the training data or by witness queries, so ω can remain partial. Likewise, the left and right automata may still separate states whose distinction no longer affects any output once the opposite side is fixed. We therefore complete the transition automata and ω, then minimize both context automata relative to the completed table.

Completing ω by representative queries. We first make each transition automaton total by adding a sink state when necessary. For each left state $q \in Q_L$ we use its representative prefix $\pi(q)$ in A_L. For each right state $p \in Q_R$ we use its representative suffix $\sigma(p)$, whose reversed read reaches p in A_R. For every still-undefined triple (q, a, p) we query the aligned output on

$$x = \pi(q)\, a\, \sigma(p).$$

By construction, the distinguished occurrence of a in this string, at position $|\pi(q)| + 1$, has left context q and right context p, so the chunk returned at that position defines $\omega(q, a, p)$. Existing output entries are left unchanged.

Partition-refinement minimization. Once ω is total, left states can be compared by the output rows they induce against *all* right states and symbols. Concretely, two left states $q_1, q_2 \in Q_L$ may be merged only if (i) $\omega(q_1, a, p) = \omega(q_2, a, p)$ for every $a \in \Sigma$ and $p \in Q_R$, and (ii) their a-successors fall into the same current equivalence classes. This is partition refinement in the style of DFA minimization (Moore, 1956; Hopcroft, 1971), but the signature of a left state is now an output row together with successor blocks. The right side is minimized symmetrically by comparing *columns* of ω across all left states. Each

Algorithm 3: AlternateMinimize($\Sigma, A_L, A_R, \omega, O; iters$)
Input: Alphabet Σ, DFAs A_L, A_R, partial or total ω, chunk oracle O (optional)
Output: Minimized bimachine (A_L, A_R, ω)

$(A_L, A_R, \omega) \leftarrow$ CompleteOmega($\Sigma, A_L, A_R, \omega, O$)
for $t \leftarrow 1$ **to** $iters$ **do**
  $(A_L, \omega) \leftarrow$ MinimizeLeftGivenRight(Σ, A_L, A_R, ω)
  $(A_L, A_R, \omega) \leftarrow$ CompleteOmega($\Sigma, A_L, A_R, \omega, O$)
  $(A_R, \omega) \leftarrow$ MinimizeRightGivenLeft(Σ, A_L, A_R, ω)
  $(A_L, A_R, \omega) \leftarrow$ CompleteOmega($\Sigma, A_L, A_R, \omega, O$)
return (A_L, A_R, ω)

quotient preserves the aligned behavior of the completed hypothesis. Moreover, quotienting identical rows preserves the equality partition of the columns, and symmetrically for quotienting columns; hence refining the left partition to stability, as in Moore's or Hopcroft's algorithm, and then the right partition to stability suffices for these quotient operations. The implementation nevertheless performs three alternating iterations, which is harmless but not required by this argument.

Remarks. Completion preserves previously defined output entries and fills missing entries using representative oracle queries. It therefore preserves previously defined runs while extending the hypothesis to additional inputs; representative queries do not certify equivalence to the teacher on all inputs. Subsequent partition-refinement minimization preserves the aligned behavior of this completed hypothesis. Operationally, it also makes the extracted machines much easier to inspect, because states whose future transition behavior and output rows or columns are indistinguishable are identified.

Implementation. The aligned learner described here is implemented in Python. In sample-only mode, witness queries are omitted. Missing output entries are filled only when the corresponding representative string occurs in the training sample, and minimization compares partial rows and columns with undefined entries treated as a distinguished value. For held-out evaluation, a position with an undefined context state or output entry receives the identity chunk (its current input symbol); agreement still requires an exact match of the entire chunk sequence. Section 4.4 reports this regime directly.

4. Experiments

We report two complementary experimental tracks. First, we evaluate the oracle-assisted aligned learner on a fixed battery of synthetic teacher oracles that return aligned chunk sequences directly.

Second, we evaluate the sample-only aligned learner on randomly generated target bimachines of various sizes and alphabet sizes, where the learner sees only aligned examples and no oracle queries are allowed during training.

For the oracle-assisted benchmark battery, the training set contains 2,000 examples with maximum input length 12, and evaluation is performed on 500 fresh random strings up to length 25. These runs use witness budget $k = 80$, continuation/head length limits 3, at most

Table 1: Aligned-regime benchmark battery. Reported sizes are the number of states in the learned left and right DFAs and the number of defined entries in ω. All tasks achieve oracle agreement 1.00 on 500 random strings of length at most 25.

Task	Alphabet / description	$\lvert Q_L\rvert$	$\lvert Q_R\rvert$	$\lvert\omega\rvert$
Swap first/last	$\Sigma = \{a, b\}$, swap first/last if $\lvert x\rvert \geq 2$	3	3	18
Future-c condition	$\Sigma = \{a, b, c\}$, $a \to b$ iff preceded by b and some c occurs to the right	2	2	12
Global begin/end	$\Sigma = \{a, b\}$, $a \to b$ iff begins and ends with b	3	3	18
Local *cad*	$\Sigma = \{a, b, c, d\}$, $a \to b$ iff in local context *cad*	2	2	16
Global-or-local	$\Sigma = \{a, b, c, d\}$, global begin/end or local *cad*	5	5	100
Parity left/right	$\Sigma = \{a, b, c\}$, parity of c to left/right	2	2	12
Even-length $a \leftrightarrow b$	$\Sigma = \{a, b, c\}$, rewrite $a \leftrightarrow b$ at every position iff $\lvert x\rvert$ is even	2	2	12
Anchored insertion	$\Sigma = \{a, b, c, d\}$, emit `ca` on the c before d	1	2	8
Conditional deletion	$\Sigma = \{a, b\}$, delete a if even length else delete b	2	2	8
Last-symbol parity deletion	$\Sigma = \{a, b\}$, delete a by parity chosen by final symbol	2	3	12

$maxRounds = 10$ merge rounds, and three alternating minimization iterations. Agreement is computed at the chunk level: the predicted chunk sequence must match the teacher's chunk sequence exactly, not just after concatenation.

4.1. Benchmark battery

We evaluated the learner on the following task families.

1. **Swap first/last (AB):** if $\lvert x\rvert \geq 2$, swap the first and last symbol; otherwise copy.
2. **Future-`c` condition (ABC):** rewrite `a`→`b` iff the symbol is immediately preceded by `b` and some `c` occurs strictly to its right.
3. **Global begin/end condition (AB):** rewrite `a`→`b` iff the word begins and ends with `b`.
4. **Local context `c a d` (ABCD):** rewrite `a`→`b` iff it is immediately preceded by `c` and immediately followed by `d`.
5. **Global-or-local (ABCD):** rewrite `a`→`b` iff the global begin/end condition holds or the local `c a d` condition holds.
6. **Parity left/right (ABC):** rewrite `a`→`b` iff the number of `c`'s to the left is even and the number to the right is odd.
7. **Even-length $a \leftrightarrow b$ (ABC):** if $\lvert x\rvert$ is even, rewrite `a`↔`b` at every position; otherwise copy.
8. **Anchored insertion (ABCD):** insert `a` between `c` and `d` by emitting chunk `ca` on the `c` before a `d`; every other symbol, including a `c` not followed by `d`, is copied.
9. **Conditional deletion (AB):** if $\lvert x\rvert$ is even delete all `a`'s, otherwise delete all `b`'s.
10. **Last-symbol parity deletion (AB):** numbering positions from 1, delete `a`'s at even positions when the last symbol is `a`, and at odd positions when the last symbol is `b`; `b`'s are always copied.

Table 2: OSTIA on the benchmark battery under unaligned, whole-output supervision. Acc. is sequence-level exact-match accuracy on the 500 test strings; $|Q|$ is the number of states of the learned subsequential transducer.

Task	Acc.	$\lvert Q\rvert$	Task	Acc.	$\lvert Q\rvert$
Swap first/last	0.370	491	Parity left/right	0.312	475
Future-c condition	0.884	41	Even-length $a \leftrightarrow b$	0.276	578
Global begin/end	0.716	188	Anchored insertion	1.000	2
Local *cad*	1.000	3	Conditional deletion	0.546	204
Global-or-local	0.742	107	Last-symbol parity deletion	0.566	221

4.2. Results

Table 1 summarizes the learned sizes. Across all ten tasks, the learner achieves perfect chunk-level oracle agreement on the 500 fresh random test strings evaluated for each task. The learned automata are compact and reflect the structural distinctions expected from the tasks: for example, the learned global-or-local bimachine has five states on each side, while the learned anchored-insertion bimachine has only one left state and two right states.

Two empirical observations are especially useful for interpreting the learner. First, output-table completion and context-automaton minimization together collapse the learned hypotheses to the compact solutions in Table 1; these experiments do not isolate an additional benefit from repeating the left/right minimization cycle. Second, the aligned learner already handles two-sided dependencies and anchored length changes directly once chunk-level alignments are supplied.

4.3. Subsequential learner baseline

For context, we evaluate OSTIA (Oncina et al., 1993) on the same battery (Table 2). OSTIA learns subsequential transducers from unaligned input–output pairs: it builds an onward prefix-tree transducer and greedily merges states, considered in lexicographic order of their access strings (Gildea and Jurafsky, 1996), redistributing output labels as needed. It receives whole outputs rather than chunk sequences and uses the same 2,000 training and 500 test inputs (seed 0, maximum lengths 12 and 25) as the aligned experiment; test inputs on which the learned transducer is undefined count as errors. The comparison thus illustrates different representation and supervision assumptions rather than controlling for the learning algorithm. Local `cad` rewriting and anchored insertion admit subsequential realizations with one-symbol delay, and OSTIA recovers compact transducers (3 and 2 states) with perfect agreement. The other eight tasks are not subsequential, since some outputs depend on an unbounded right context. OSTIA's hypotheses fit the sample but none reaches perfect agreement, and several grow to hundreds of states; the higher scores (e.g. 0.884 on future-c) therefore reflect partial approximation, not recovery of the target.

4.4. Sample-only recovery on random bimachines

The oracle-assisted battery above establishes that the implemented aligned learner can recover compact bimachines when witness queries are available. To understand how much

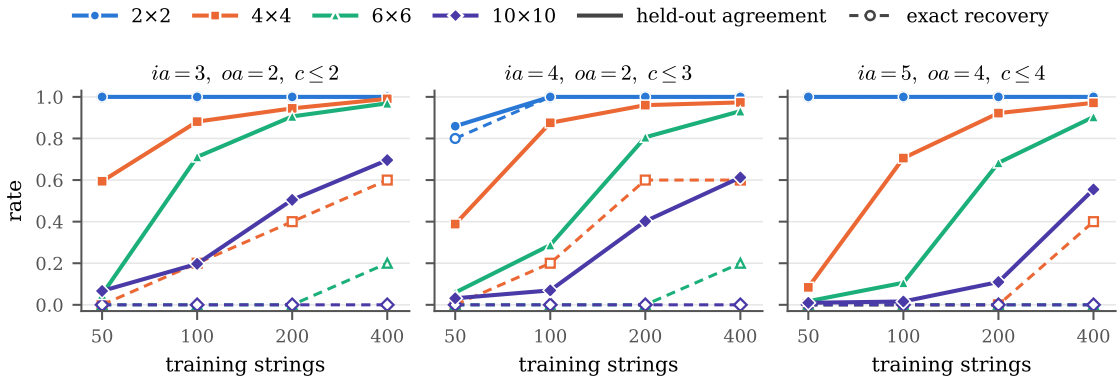


Figure 3: Sample-only recovery on random target bimachines, one panel per family (input alphabet size ia, output alphabet size oa, maximum chunk length c). Color and marker give the target size. Solid lines report mean held-out agreement and dashed lines exact normalized recovery, over five targets; where the two coincide (e.g. 2×2 at 1.0) the dashed line is hidden under the solid one. The horizontal axis is logarithmic.

of this behavior survives without oracle access, we next evaluate the sample-only learner on randomly generated target bimachines.

Each target is a random total bimachine: transition targets are sampled independently and uniformly from the states on the corresponding side, and targets with unreachable states are rejected. Chunk lengths are uniform on $\{0, \ldots, c\}$ and their symbols are independent and uniform over an output alphabet disjoint from Σ. We apply the prescribed minimization and deterministic breadth-first renumbering, rejecting targets if either side shrinks below the requested size. This is an operational normalization, not a canonical form for the unaligned string function: different chunk allocations may realize the same function.

For each target and $n \in \{50, 100, 200, 400\}$, distinct training strings have uniform lengths in $\{0, \ldots, 6\}$ and independent uniform symbols. Evaluation uses 700 analogously generated strings of lengths in $\{0, \ldots, 10\}$ that do not occur in the training sample, with the target used only for evaluation. *Held-out agreement* is exact chunk-sequence agreement on these test strings; *exact normalized recovery* directly compares both transition tables and all defined output entries after minimization and deterministic renumbering. All 240 runs (three families, four target sizes, four sample sizes, five targets each) completed with train fit 1.0; five targets per point make recovery rates multiples of 0.2. In Figure 3 and Table 3, 'ia' denotes input alphabet size, 'oa' output alphabet size, and 'c' the maximum output-chunk length used when generating the target bimachines.

Figure 3 shows a clear sample-complexity gradient. Exact recovery emerges quickly on the smallest targets: the 2×2 targets are recovered exactly with 50 examples in two families and with 100 in the third (ia=4). For moderate and larger targets, held-out agreement rises well before exact normalized recovery, showing that sample-only learning can generalize without reconstructing the target exactly.

Table 3: Sample-only random-target summary. Entries are *held-out agreement / exact normalized recovery* for representative target sizes, averaged over five random targets.

Family	Size	$n_{\text{train}} = 50$	$n_{\text{train}} = 100$	$n_{\text{train}} = 200$	$n_{\text{train}} = 400$
ia=3, oa=2, $c\leq$2	4 × 4	0.594/0.0	0.881/0.2	0.945/0.4	0.991/0.6
ia=3, oa=2, $c\leq$2	6 × 6	0.050/0.0	0.711/0.0	0.906/0.0	0.969/0.2
ia=3, oa=2, $c\leq$2	10 × 10	0.067/0.0	0.197/0.0	0.505/0.0	0.696/0.0
ia=4, oa=2, $c\leq$3	4 × 4	0.388/0.0	0.875/0.2	0.960/0.6	0.973/0.6
ia=4, oa=2, $c\leq$3	6 × 6	0.059/0.0	0.288/0.0	0.806/0.0	0.931/0.2
ia=4, oa=2, $c\leq$3	10 × 10	0.031/0.0	0.069/0.0	0.402/0.0	0.613/0.0
ia=5, oa=4, $c\leq$4	4 × 4	0.084/0.0	0.706/0.0	0.922/0.0	0.971/0.4
ia=5, oa=4, $c\leq$4	6 × 6	0.017/0.0	0.106/0.0	0.683/0.0	0.904/0.0
ia=5, oa=4, $c\leq$4	10 × 10	0.010/0.0	0.016/0.0	0.110/0.0	0.555/0.0

Table 3 quantifies this separation: for example, at ia=4, oa=2, $c\leq$3, the 4 × 4 targets reach mean held-out agreement 0.875 with 100 examples, while exact recovery is only 0.2. This is the tradeoff that the no-oracle evaluation is meant to expose. Test strings that reach an undefined state or output entry receive the identity fallback, whose chunk (an input symbol) never occurs in the disjoint output alphabet and is therefore always wrong; at $n = 50$, 76–85% of test strings on 6 × 6 and 10 × 10 targets fall in this case. The length cap also limits exact recovery independently of n: strings of length at most 6 reach only 81–96% of the output entries of each 10 × 10 target with ia=3, 98–99% with ia=4, and 97–98% for two of the five ia=5 targets (and 97% for one 6 × 6 target with ia=4), so those targets cannot be recovered exactly from any such sample.

5. Discussion and conclusion

We have described a practical, stand-alone bimachine learner for chunk-labeled functional transductions. It supports deletions and anchored insertions and recovers compact hypotheses with perfect chunk-level agreement on 500 random test strings of length at most 25 for each local, global, and two-sided oracle benchmark. In the strictly sample-only regime, smaller random targets can often be recovered exactly, while larger ones show a clear sample-complexity tradeoff in held-out agreement before exact structural recovery.

Future work includes learning the chunk alignment itself from unaligned input-output samples, rather than assuming it as a supervision signal, as well as distilling neural translation models into symbolic bimachines.

Bidirectional recurrent encoders (Schuster and Paliwal, 1997), including those used in attention-based translation (Bahdanau et al., 2015), are natural candidates because their positionwise representations combine left and right context; the analogy does not extend to autoregressive decoders, whose outputs need not arise from a fixed local table.

References

Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. Neural machine translation by jointly learning to align and translate. In *3rd International Conference on Learning Representations (ICLR)*, 2015. URL https://arxiv.org/abs/1409.0473.

Adrien Boiret, Aurélien Lemay, and Joachim Niehren. Learning rational functions. In *Developments in Language Theory*, volume 7410 of *Lecture Notes in Computer Science*, pages 273–283. Springer, 2012. doi: 10.1007/978-3-642-31653-1_25.

Colin de la Higuera. *Grammatical Inference: Learning Automata and Grammars*. Cambridge University Press, Cambridge, 2010.

Daniel Gildea and Daniel Jurafsky. Learning bias and phonological-rule induction. *Computational Linguistics*, 22(4):497–530, 1996.

John Hopcroft. An n log n algorithm for minimizing states in a finite automaton. In *Theory of machines and computations*, pages 189–196. Elsevier, 1971.

Ronald M. Kaplan and Martin Kay. Regular models of phonological rule systems. *Computational Linguistics*, 20(3):331–378, 1994.

Kevin J. Lang, Barak A. Pearlmutter, and Rodney A. Price. Results of the Abbadingo One DFA learning competition and a new evidence-driven state merging algorithm. In *Grammatical Inference (ICGI 1998)*, volume 1433 of *Lecture Notes in Computer Science*, pages 1–12. Springer, 1998.

Mehryar Mohri. Finite-state transducers in language and speech processing. *Computational Linguistics*, 23(2):269–311, 1997.

Edward F. Moore. Gedanken-experiments on sequential machines. In Claude E. Shannon and John McCarthy, editors, *Automata Studies*, number 34 in Annals of Mathematics Studies, pages 129–153. Princeton University Press, 1956.

José Oncina and Pedro García. Inferring regular languages in polynomial updated time. In *Pattern Recognition and Image Analysis*, pages 49–61. World Scientific, 1992. doi: 10.1142/9789812797902_0004.

José Oncina, Pedro García, and Enrique Vidal. Learning subsequential transducers for pattern recognition interpretation tasks. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 15(5):448–458, 1993. doi: 10.1109/34.211465.

Christophe Reutenauer and Marcel-Paul Schützenberger. Minimization of rational word functions. *SIAM Journal on Computing*, 20(4):669–685, 1991.

Emmanuel Roche. Compact factorization of finite-state transducers and finite-state automata. *Nordic Journal of Computing*, 4(2):187–216, 1997.

Mike Schuster and Kuldip K. Paliwal. Bidirectional recurrent neural networks. *IEEE Transactions on Signal Processing*, 45(11):2673–2681, 1997. doi: 10.1109/78.650093.

Marcel-Paul Schützenberger. A remark on finite transducers. *Information and Control*, 4 (2–3):185–196, 1961.

Wojciech Skut, Stefan Ulrich, and Kathrine Hammervold. A bimachine compiler for ranked tagging rules. In *Proceedings of the 20th International Conference on Computational Linguistics*, pages 198–204, 2004. doi: 10.3115/1220355.1220384.