# PROVABLE CHECKPOINT REUSE FOR REGMIX PROXY SWARMS

**Anonymous authors**
Paper under double-blind review

## ABSTRACT

The composition of pre-training data is a primary determinant of language-model quality, which makes mixture optimization a standard stage of large-scale training pipelines. Existing regression-based mixture search methods pioneered by a RegMix-style pipeline fit a surrogate to a swarm of proxy-model runs and therefore train many candidate mixtures from scratch even though their cumulative compositions may overlap heavily. Therefore, we introduce the Mixture Training Reuse (MTR) problem, which aims to realize a fixed family of target data mixtures at minimum training cost by treating every target mixture as a cumulative process to its terminal composition and sharing intermediate checkpoints across runs. For the arbitrary-order problem MTR-a where the order of training examples does not matter, we show that the best shared checkpoint for a set of targets is their coordinatewise minimum, which reduces the continuous checkpoint search to laminar optimization over target subsets; building on this reduction, we prove NP-completeness and a tight $1/2$-approximation reuse-gain guarantee for a greedy agglomeration algorithm. For the random-order problem MTR-r, we design a descendant-aware greedy planner that admits a checkpoint only when every descendant target satisfies a feasibility test measured by total variation (TV). Our evaluation covers both synthetic mixture families and real-data training. On tractable synthetic instances, `MTR-a` recovers $99.49\%$ of the training savings attained by the exact dynamic-programming optimum ($49.09\%$ versus $49.34\%$). On a 128-mixture Essential-Web proxy swarm, `MTR-a` saves $70.0\%$ of training tokens, while TV-constrained `MTR-r` saves $49.0\%$ with a closer compositional match to independent training.

## 1 INTRODUCTION

Large language models are trained on corpora from heterogeneous sources. In modern pre-training and post-training pipelines, a fixed training budget is allocated across multiple source types (e.g., math, code, books, etc.) with carefully chosen mixture weights. This data-mixture choice is a first-class design decision (Chen et al., 2026b), because it mediates the tradeoff between broad coverage and targeted capability.

Recent work has made mixture selection increasingly systematic by treating it as an optimization problem rather than an ad hoc heuristic (Xie et al., 2023; Fan et al., 2024). RegMix (Liu et al., 2025) pioneers a prominent paradigm for mixture selection that has been validated in industrial-grade LLM training (Team et al., 2025): it trains a swarm of proxy models under various candidate mixtures, fits a surrogate from mixture weights to their downstream performance, and finally obtains an improved mixture from the surrogate. The *Olmix* framework extends this perspective to the full development cycle, emphasizing careful configuration of the offline schema and mixture reuse when the domain set evolves (Chen et al., 2026a). CLIMB pushes the same agenda toward less curated corpora by clustering raw data into induced domains and iteratively bootstrapping better mixtures over those clusters (Diao et al., 2025). Taken together, these methods cast data mixing as an iterative computational pipeline whose inner loop repeatedly trains models on many candidate mixtures.

The computational burden in that pipeline is to realize a large family of candidate target mixtures, $P = \{p^{(1)}, \ldots, p^{(m)}\}$. Here each target $p^{(j)}$ is a mixture vector whose coordinates specify the fraction of the training budget allocated to each source. Swarms, ablations, and iterative updates



Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

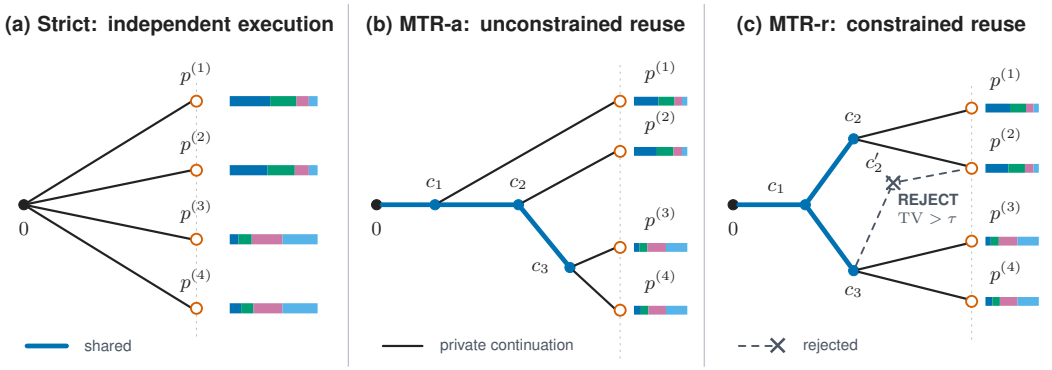



Figure 1: Three reuse plans for the same four target mixtures $p^{(1)}, \ldots, p^{(4)}$, each being a composition strip over the source domains. Each plan is a rooted tree whose internal nodes are checkpoints, and whose leaves are the targets. (*a*) The independent baseline trains every target separately, so all edges are private continuations. (*b*) MTR-a routes the targets through shared checkpoints $c_1, c_2, c_3$: targets that agree on an initial allocation can train the shared prefix once and diverge only along target-specific tails. (*c*) MTR-r is similar to MTR-a but additionally enforces a feasibility test that measures the deviation of shared edges from each target mixture.

routinely require training many models whose target mixtures may overlap substantially. Treating each target independently discards this shared structure and repeats work that could in principle be amortized through checkpoint reuse. For example, if many candidates share the same $80\%$ partial allocation to a set of sources and differ only in the remaining $20\%$, then one can train that shared partial allocation once and branch to complete each candidate. This motivates the central question of this paper: given a target set $P$, how should one organize reusable checkpoints so that every target mixture is still reached exactly, while the total amount of training is minimized?

To make this question precise, we introduce the Mixture Training Reuse (MTR) problem and study it under two different settings. First, we consider an arbitrary-order setting (MTR-a), in which the order of training data is irrelevant and only the cumulative amount consumed from each source matters. We show that the optimal checkpoint for a fixed set of targets is the coordinatewise minimum of those targets. This identity reduces the continuous checkpoint search into a laminar combinatorial optimization problem over subsets of targets. Building on that structure, we prove NP-hardness and analyze a greedy agglomerative algorithm that repeatedly merges the pair of current blocks of targets $A$ and $B$ whose union yields the largest reuse gain, and we show that it provably attains *at least half of the optimal reuse gain* for MTR-a.

In the second random-order setting (MTR-r), we further impose additional constraints on each training trajectory so that data from different sources are sampled in a more balanced and randomized order, with no source type systematically frontloaded or deferred. These constraints matter because bounding the composition along every reuse trajectory leads to a more faithful approximation to their independent training. We therefore formalize MTR-r as the constrained counterpart of MTR-a, in which a merge is feasible only if every descendant target satisfies a test based on total variation (TV). We devise a similar but descendant-aware greedy planner that repeatedly selects the feasible merge with the largest reuse gain, and verify its effectiveness in practice. Figure 1 illustrates the independent baseline and the plans produced by the two settings for the same family of four targets.

The remainder of this paper is organized as follows. Section 2 reviews related work, and Section 3 formalizes the MTR problem and its two settings. Sections 4 and 5 analyze the arbitrary-order setting and present a provable greedy planner, Section 6 develops the planner for the random-order setting, and Section 7 evaluates both before Section 8 concludes. All proofs are provided in Appendix A.

## 2 Related Work

Mixture optimization for large-scale pre-training divides into offline search, which selects a fixed mixture before the principal training run, and online reweighting, which adjusts domain weights

during training. The dominant offline pipeline trains proxy models for candidate mixtures and fits a surrogate that predicts the performance of unseen mixtures, as in DoReMi (Xie et al., 2023), RegMix (Liu et al., 2025), and the systematized workflow of *Olmix* (Chen et al., 2026a); a broader survey of this family is provided by Chen et al. (2026b). Online methods instead optimize a single evolving data trajectory, for example through per-domain scaling laws (Jiang et al., 2025) or an online-estimated mixing law (Chen et al., 2025). A parallel line reduces search cost by training domain-specific parameter components and merging them to approximate candidate mixtures (Li et al., 2026; Wang et al., 2026b; Hu et al., 2026), although such merging may not capture cross-domain interactions accurately. Finally, the transferability of proxy-based conclusions across scales, configurations, and horizons remains an active question (Ye et al., 2025; Khaddaj et al., 2025; Wang et al., 2026a; Gu et al., 2025; Zhou et al., 2026). MTR is complementary to all of these methods: it treats the candidate mixtures as given and optimizes how their training runs share intermediate checkpoints. Appendix B provides an expanded discussion.

## 3 Problem formulation

Suppose the training data are partitioned into $d$ source domains. A target mixture is a vector $p \in \Delta^{d-1} = \left\{p \in \mathbb{R}^d_{\geq 0} : \sum_{i=1}^d p_i = 1\right\}$, where we normalize the training budget of each target to one unit. We are given a collection of target mixtures $P = \left\{p^{(1)}, \ldots, p^{(m)}\right\} \subseteq \Delta^{d-1}$. Each target is trained from scratch for the same total budget (e.g., a fixed number of tokens). Under this normalization, the training trajectory of a target $j$ corresponds to a continuous process from vector $0$ to its terminal vector $p^{(j)}$. A checkpoint $c \in [0,1]^d$ records the cumulative amount consumed from each domain in this process, where $c_i \leq p_i^{(j)}$ is the amount of the $i$-th source domain.

The simplest approach to realize all target mixtures in $P$ trains every target independently, and pays the full training budget for each target. Under the unit-budget normalization, training a single target from scratch costs exactly $\left\|p^{(j)}\right\|_1 = 1$, so this independent baseline pays a total cost of $m$.

This simple approach is wasteful, because the targets in $P$ typically share a large amount of structure: proxy swarms routinely contain many mixtures that agree on their allocation on most domains and differ only in a small remainder, and training each of them from scratch redoes the same work unnecessarily. Therefore, we propose to reuse checkpoints to remove the duplication, since a group of targets that agrees on an initial allocation $q \in [0,1]^d$ can train the shared prefix $q$ once and then diverge only along target-specific tails. So a well-chosen reuse plan can drive the total training cost far below the baseline $m$.

We formalize such a reuse plan as a rooted directed tree $T$ whose nodes are checkpoints and whose edges represent continued training. The root is labeled by the zero checkpoint $0$, and the leaves are in bijection with the targets in $P$, so that leaf $j$ is labeled by $p^{(j)}$. Every edge $(u, v)$ is *monotone*, meaning $c(u) \leq c(v)$ coordinatewise, so that traversing it corresponds to continuing training rather than rewinding. The training budget spent on an edge is its $\ell_1$ increment $\|c(v) - c(u)\|_1$, and the cost of the plan $\mathrm{cost}(T)$ thus equals the total sum of training budget over all edges. That is, $\mathrm{cost}(T) = \sum_{(u,v)\in E(T)} \|c(v) - c(u)\|_1$. Note that because $T$ is a tree, a plan can be executed by a depth-first traversal, and only the checkpoints on the current root-to-leaf path need to be materialized at any time.

The problem we study is therefore to realize every target in $P$ exactly while minimizing the total amount of training, which amounts to organizing reusable checkpoints as a rooted tree or forest $T$ under certain constraints so that as much training as possible is paid for once and shared, i.e., $\mathrm{cost}(T)$ is minimized. We describe the constraints for two different settings below. Figure 1 illustrates the different reuse plans, and highlights the feasible shared-prefix and branching decisions made under different constraints.

**Arbitrary-order setting.** In the arbitrary-order idealization, the order of training examples does not matter; only the cumulative amount from each source matters to realize a target mixture.

**Problem 1** (Mixture Training Reuse Problem, arbitrary-order version (MTR-a))**.** *Given the target set $P$, construct a rooted directed tree $T$ with root checkpoint $0$, leaves bijective with the targets $P$ and monotone edges. The goal is to minimize* $\mathrm{cost}(T)$.

**Random-order setting.** The arbitrary-order idealization may be too permissive to approximate the independent baseline faithfully in some cases. The random-order setting instead requires the sources of every reuse trajectory to be consumed in a more balanced and randomized order, so that no source domain is systematically frontloaded or deferred. We therefore attach to a reuse plan an abstract feasibility test $\mathcal{F}$ that rejects plans whose root-to-leaf trajectories deviate too much from training the corresponding targets independently. We will instantiate $\mathcal{F}$ with concrete trajectory tests in Section 6.

**Problem 2** (Mixture Training Reuse Problem, random-order version (MTR-r))**.** *Given the target set $P$ and a feasibility test $\mathcal{F}$, construct a rooted directed tree $T$ with root checkpoint $0$, leaves bijective with the targets $P$, and monotone edges, such that $T$ satisfies $\mathcal{F}$. The goal is to minimize* $\mathrm{cost}(T)$.

# 4 MTR-A IS COMBINATORIAL AND NP-HARD

We now examine the structure of the arbitrary-order problem and ask how difficult it is to solve optimally. The analysis rests on a single observation: for any set of targets, the best shared checkpoint is the coordinatewise minimum of the targets it serves, because this checkpoint is the largest prefix that all of them can share without violating feasibility. This identity avoids the continuous search over checkpoint vectors and leaves us with a combinatorial choice of how descendant sets are nested. Section 4.1 develops this choice into an equivalent laminar optimization problem. Building on that reformulation, Section 4.2 shows that the problem is genuinely hard by proving that MTR-a is NP-complete, via a reduction from triangle decomposition. This hardness result explains why we turn to a greedy approximation in Section 5.

## 4.1 LAMINAR REFORMULATION

MTR-a can be reformulated as a purely combinatorial optimization problem over laminar families of terminal subsets.

For every non-empty subset $S \subseteq [m]$, define its *maximal common checkpoint* $q(S)$ by $q(S)_i = \min_{j \in S} p_i^{(j)}$ for $i = 1, \ldots, d$, and its *overlap weight* $w(S) = \|q(S)\|_1$, the total training mass that every target in $S$ can share. The best shared checkpoint is the coordinatewise minimum of its targets.

**Lemma 1** (Best checkpoint for a descendant set)**.** *Fix a feasible tree topology. For every internal node $u$ with descendant leaf set $S(u)$, the cost is minimized by the checkpoint $c(u) = q(S(u))$.*

By Lemma 1, any feasible MTR-a tree can be checkpoint-relabeled without increasing the cost, so the only remaining decision is the topology, or equivalently, which descendant sets appear in it. The descendant sets of two internal nodes are nested when one is an ancestor of the other and disjoint otherwise, so they form a laminar family $\mathcal{L}$, i.e., every pair $A, B \in \mathcal{L}$ satisfies $A \subseteq B$, $B \subseteq A$, or $A \cap B = \varnothing$. For a laminar family $\mathcal{L}$ over $[m]$ that contains $[m]$ and every singleton $\{i\}$, containment turns $\mathcal{L}$ into a rooted tree whose root is $[m]$ and whose leaves are the singletons. Writing $\mathrm{ch}_{\mathcal{L}}(S)$ for the number of maximal proper subsets of $S$ in $\mathcal{L}$, i.e., its number of children, and $U(\mathcal{L}) = \{S \in \mathcal{L} : S \neq [m],\ |S| \geq 2\}$ for the family of non-root internal sets, we define the reuse gain of $\mathcal{L}$ by

$$\Phi(\mathcal{L}) = \sum_{S \in U(\mathcal{L})} \big(\mathrm{ch}_{\mathcal{L}}(S) - 1\big) w(S). \tag{1}$$

**Theorem 1** (Equivalence of MTR-a and laminar tree selection)**.** *Maximizing $\Phi(\mathcal{L})$ over laminar families $\mathcal{L}$ containing $[m]$ and every singleton is equivalent to MTR-a: every checkpoint-relabeled MTR-a tree $T$ admits such a family with $\mathrm{cost}(T) = m - \Phi(\mathcal{L})$, and conversely every such family $\mathcal{L}$ is realized by a feasible MTR-a tree $T$ with $\mathrm{cost}(T) = m - \Phi(\mathcal{L})$. Consequently, $\min_T \mathrm{cost}(T) = m - \max_{\mathcal{L}} \Phi(\mathcal{L})$, so the two problems are equivalent.*

## 4.2 COMPUTATIONAL HARDNESS

We give a reduction showing that the MTR-a problem is NP-hard. We reduce from *triangle decomposition*: given a simple undirected graph $G = (V, E)$ with $|E| = 3r$, decide whether $E$ can be partitioned into $r$ edge-disjoint triangles. This problem is NP-complete (Dor & Tarsi, 1997).

The construction assigns one target to each edge of the input graph. The only shared mass between different targets is placed on coordinates indexed by triangles, so an internal checkpoint yields a positive reuse gain exactly when all of its descendant edges lie in a common triangle. For example, if $A = \{e_1, e_2, e_3\}$ is a triangle, then checkpoints containing edges $\{e_1, e_2\}$, $\{e_1, e_3\}$, $\{e_2, e_3\}$ or $\{e_1, e_2, e_3\}$ are exactly the vector that has a positive value on the coordinate indexed by $A$ and a value 0 on every other coordinate. Then we show that the maximum reuse gain is attained only when all $3r$ edges are partitioned into $r$ disjoint three-edge groups, each corresponding to a triangle, which is equivalent to a triangle decomposition.

**Theorem 2** (NP-hardness of MTR-a)**.** *The decision version of MTR-a is NP-complete.*

## 5 GREEDY AGGLOMERATION FOR MTR-A

In this section, we propose and analyze a greedy agglomerative algorithm for the MTR-a problem, which repeatedly merges two current groups $A$ and $B$ of targets into a new checkpoint $q(A \cup B)$ that maximizes the shared overlap weight $w(A \cup B)$, until a single tree is formed. We show that this algorithm provably attains at least half of the optimal reuse gain for MTR-a. We further show that the approximation factor $1/2$ is tight.

### 5.1 1/2-APPROXIMATION GUARANTEE ON REUSE SAVINGS

For every nonempty set $S \subseteq [m]$, recall that $q(S) \in [0,1]^d$ is the coordinatewise minimum checkpoint and $w(S) = \|q(S)\|_1$ is its overlap weight. For any feasible MTR-a tree $T$, define the reuse gain by $\Phi(T) = m - \mathrm{cost}(T)$. Therefore maximizing $\Phi(T)$ is equivalent to minimizing $\mathrm{cost}(T)$.

We start the greedy agglomeration from the singleton partition of $[m]$. At each step we choose two current blocks $A$ and $B$ maximizing $w(A \cup B)$ and replace them by a merged block $S = A \cup B$. Let $S_1, \ldots, S_{m-1}$ be the successive merged blocks for in total $m-1$ steps, and let $g_t = w(S_t)$ be their merge weights. We construct a feasible MTR-a tree $T_{\mathrm{g}}$ by adding one internal node for each merged block $S_t$ labeled by $q(S_t)$ and finally introducing a structural root labeled by the zero checkpoint. We attach children according to the merge history, so that each internal node has exactly two children corresponding to the two blocks merged at that step. We also attach the unique internal node corresponding to $S_{m-1} = [m]$ as the unique child of the structural root, so that $[m]$ is represented by a non-root internal node and hence lies in $U(T_{\mathrm{g}})$. This convention simplifies the subsequent analysis by ensuring that the $m-1$ greedy merge steps are in bijection with the $m-1$ nodes of $U(T_{\mathrm{g}})$, and also ensuring that $\Phi(T_{\mathrm{g}}) = \sum_{t=1}^{m-1} g_t$. Figure 2 illustrates this construction on $m = 4$ targets, where the greedy rule first merges the two pairs $\{1,2\}$ and $\{3,4\}$ and then merges the resulting blocks.

Next, we state the main theorem, which shows the effectiveness of the above greedy algorithm: it always attains at least half of the optimal reuse gain, even in the worst case.

**Theorem 3.** *Let $T_{\mathrm{g}}$ be the greedy agglomeration tree defined above, and let $T^\star$ be an optimal MTR-a tree. Then $\Phi(T_{\mathrm{g}}) \geq \frac{1}{2}\Phi(T^\star)$, and equivalently $\mathrm{cost}(T_{\mathrm{g}}) \leq \frac{1}{2}\big(m + \mathrm{cost}(T^\star)\big)$.*

The key intuition behind the guarantee is to view $w(S)$ as the amount of computation that can be shared by all targets in $S$. For instance, recall $\Phi(T_{\mathrm{g}}) = \sum_{t=1}^{m-1} g_t$. Thus, it is enough to examine in $U(T_{\mathrm{g}})$ and $U(T^\star)$, for each threshold $\tau \in [0,1]$, their numbers of merge points whose shared computation exceeds $\tau$. Then, a simple yet crucial monotonicity property of $w$ (see Lemma 2 below) under set inclusion then restricts the optimal tree $T^\star$ to create no more than twice as many such useful merge points as the greedy tree for each threshold $\tau$. Summing this comparison over all thresholds gives the 1/2-approximation guarantee.

**Lemma 2.** *If $A \subseteq B$ are nonempty sets, then $w(A) \geq w(B)$.*

**Fast implementation.** A direct implementation of the greedy rule costs $O(m^3 d)$ time, which is prohibitive for a large number of targets $m$. An exact priority-queue implementation reduces this to $O(m^2(d + \log m))$ time and $O(m^2)$ memory, and for very large $m$ we restrict merge proposals to approximate nearest neighbors of the current block checkpoints under an $\ell_1$ proxy, which yields an approximately greedy variant with $O\big(m(\tau + d + \log m)\big)$ time where $\tau$ is the cost of one approximate-nearest-neighbor query. Appendix C details both implementations and the proposal rule.

### 5.2 TIGHTNESS OF THE APPROXIMATION FACTOR $1/2$

We further show that the approximation factor $1/2$ in Theorem 3 is essentially the best possible for this greedy merge rule. The following theorem exhibits a family of MTR-a instances on which greedy agglomeration is forced to commit to a locally optimal but globally suboptimal first merge, after which no further reuse can be obtained. One such instance is summarized in Figure 3. This also confirms the fact that the analysis in Theorem 3 cannot be strengthened and the approximation factor $1/2$ is tight in the worst case.

**Theorem 4.** *For every $\varepsilon > 0$, there exists an MTR-a instance for which the greedy agglomeration tree $T_{\mathrm{g}}$ satisfies*

$$\Phi(T_{\mathrm{g}}) \leq \left(\tfrac{1}{2} + \varepsilon\right)\Phi(T^{\star}),$$

*where $T^{\star}$ is an optimal MTR-a tree.*

## 6 MTR-R WITH FEASIBILITY CONSTRAINTS

MTR-a maximizes shared cumulative training mass but does not constrain the order in which domains are exposed. A shared prefix that frontloads one domain and defers another is thus as acceptable to MTR-a as the random-order one that the target mixture is meant to approximate. MTR-r addresses this limitation by restricting feasible reuse structures with a cumulative Forward-TV test. The test measures the deviation of each training edge from a target mixture by a *bounded* total variation (TV) between two normalized mixtures. For a descendant target $j$, it aggregates these deviations along the root-to-leaf path into a score $F_j$, and it rejects a candidate merge as soon as some descendant violates $F_j \leq \tau_f$. The cap $\tau_f$ is user-specified. The following paragraphs define $F_j$ precisely and incorporate this feasibility test into the greedy planner.

**Forward-TV.** An edge of the reuse tree represents continued training from a parent checkpoint to a child checkpoint, and its increment $\Delta$ records how much of each source domain is consumed on that edge; the training budget spent on the edge is its length $\ell = \|\Delta\|_1$. If $\ell > 0$, the edge realizes a normalized mixture $r = \Delta/\ell$, and the deviation of this realized mixture $r$ from a target mixture $p$ is measured by their total variation, $\mathrm{TV}(r, p) = \frac{1}{2}\|r - p\|_1$. Charging each edge in proportion to the budget that it costs then yields the edge contribution $L_{\mathrm{fwd}}(\Delta, p) = \ell\,\mathrm{TV}(r, p)$, which is zero when $\ell = 0$, because such an edge trains nothing. The Forward-TV score of target $j$ is the sum of these contributions over all edges on its root-to-leaf path, $F_j = \sum_e L_{\mathrm{fwd}}(\Delta_e, p^{(j)})$, where the sum ranges over the edges of the root-to-leaf path of $j$. The score is therefore large when a target spends much of its budget on segments whose mixture departs from $p^{(j)}$, and it vanishes when every segment reproduces $p^{(j)}$. Note that as the reuse tree is being constructed, when a new parent checkpoint $a$ is inserted above an existing block root $b$, the old edge $(0, b)$ from the zero checkpoint to $b$ is replaced by two edges $(0, a)$ and $(a, b)$, and the score is updated accordingly. Figure 4 traces this update for a single target.

**Best feasible checkpoint for a fixed pair as a linear program.** Constrained by the above feasibility test, finding the best feasible checkpoint for a fixed pair of target blocks becomes harder than Lemma 1, because a feasible checkpoint must satisfy the Forward-TV cap for every descendant of both blocks. The nonsmooth TV that defines $L_{\mathrm{fwd}}$ nevertheless admits a linear representation, so the constrained search remains a linear program whose optimum maximizes the shared mass of the pair. Appendix D proves this representation and states the program as Theorem 5. The exact LP is useful as an oracle or as a refinement step for a shortlist of promising pairs, while solving it for every pair can be computationally prohibitive. This observation motivates the one-dimensional diagonal restriction below.

**Greedy construction with the diagonal search.** The practical implementation replaces the pairwise LP by the candidate checkpoint family $\alpha q(A \cup B)$, where $\alpha \in \{1, 0.95, \ldots, 0.10\}$. This family uses one scalar $\alpha$ to scale the largest coordinatewise common checkpoint, so it is a conservative but efficient diagonal approximation to the LP feasible region. For every pair of current blocks, the planner enumerates the grid candidates in decreasing order of $\alpha$ and tests each with the full Forward-TV score above, so the first feasible candidate is the one with the largest feasible $\alpha$ and the largest shared mass $\|q\|_1$ that can be realized. Note that it is possible that no feasible candidate exists for a pair.

**Algorithm 1:** Greedy MTR-r with a Forward-TV constraint (diagonal implementation).

**Input:** Target mixtures $P$ and Forward-TV cap $\tau_f$.
**Output:** A feasible reuse tree/forest.

```
Initialize one block per target with checkpoint p^(j) ∈ P and score b_j^{j} = 0
while a feasible merge exists do
    for each pair of current blocks (A, B) do
        for α = 1, 0.95, ..., 0.10 do
            if α q(A ∪ B) satisfies the Forward-TV cap for every descendant then
                Record α q(A ∪ B) as the candidate of (A, B) and break
    Merge the pair (A, B) with the candidate whose shared mass ‖q‖_1 is the largest
return the reuse tree/forest
```

Then, similar to the greedy planner for MTR-a, the merge executed next is the feasible candidate with the largest shared mass over all pairs. The loop terminates as soon as no pair admits a feasible candidate any more. The complete algorithm is displayed in Algorithm 1 with $O\big(|G|m^3d\big)$ time and $O(md)$ memory, where $G$ is the set of $\alpha$ values; details are in Appendix C.1.

## 7 Experiments

We evaluate MTR through planner-level controlled experiments on synthetic mixture families and end-to-end proxy training on 128 proxy-training mixtures constructed from Essential-Web v1.0 (Essential AI et al., 2025), with approximately 1B training tokens allocated to each proxy. Our evaluation asks how much MTR reduces proxy-training cost through checkpoint reuse. It also measures how closely MTR preserves the final selected composition and proxy-loss ranking obtained through independent training. Additional results are deferred to the appendix, including the complete Forward-TV threshold sweeps in Appendix E.4, and the ANN-based planner scalability study in Appendix E.7.

**Baselines.** We consider four reuse strategies, forming a progression from independent training to constrained hierarchical reuse. `Strict` trains every target mixture independently from the same initial weights without checkpoint reuse. `Star` trains the largest single checkpoint shared by all targets before forking independent tails and serves as the one-level reuse reference. `MTR-a` builds a reuse tree through greedy pairwise merging, as described in Section 5. `MTR-r` uses the same greedy tree-construction framework while filtering candidate merges whose pathwise Forward-TV scores exceed a threshold $\tau_f$, to keep the shared prefixes aligned with their target compositions; see Section 6. We use $\tau_f = 0.20$ as the default operating point.

**Metrics.** Computational efficiency is measured using token savings and GPU-hours. Token savings quantify the fraction of training tokens avoided relative to `Strict`'s independent training. GPU-hours sum the wall-clock execution time across all GPU workers, including model training and the checkpoint save/load operations required by MTR.

Beyond computational cost, we assess whether MTR preserves the outcome of independent training through the final selected composition and the ordering of the observed proxy losses. Following the RegMix protocol (Liu et al., 2025), each fitted surrogate selects the top-$k$ lowest-predicted-loss mixtures $S_k$ from a shared pool of 100,000 candidates, whose mean defines the final domain composition $\hat{p}_k$, i.e., $\hat{p}_k = \frac{1}{k}\sum_{i\in S_k} p_i$. For the former, we compare this composition with the composition produced by `Strict` under independent training, using the $L_1$ distance and maximum domain deviation, $\|\hat{p}_k^{(q)} - \hat{p}_k^{(\mathrm{Strict})}\|_1$ and $\|\hat{p}_k^{(q)} - \hat{p}_k^{(\mathrm{Strict})}\|_\infty$, where $q$ denotes an execution strategy. Regarding the latter, we report proxy Spearman that measures rank correlation between the top-$k$ observed proxy losses under a reuse strategy and its paired `Strict` run.

### 7.1 CONTROLLED EXPERIMENTS ON SYNTHETIC MIXTURES

The synthetic study isolates the topology-planning behavior of the evaluated reuse strategies under controlled mixture geometries. We construct eleven mixture families that span symmetric, peaked, sparse, clustered, one-hot, and RegMix-like weight structures. We also vary $m$ and $d$ respectively while holding the other parameter fixed. The complete distribution family definitions and per-family results are in Appendix E.6.

**Near-optimal greedy planning.** As established by the NP-hardness result in Theorem 2, computing the globally optimal reuse tree is intractable at realistic scales. We therefore obtain exact references through dynamic programming (DP) on small instances (Table 1). The benchmark spans $m \in \{5, 8, 10, 12\}$ and $d \in \{4, 8\}$ over 20 random seeds per configuration. On these instances, `MTR-a` achieves $49.09\%$ average token savings, nearly matching the $49.34\%$ exact optimum, while `MTR-r` retains $36.10\%$ savings. Both far exceed the $16.00\%$ by `Star`. For a like-for-like comparison, `MTR-r` also remains close to the exact Forward-TV-constrained DP optimum, as detailed in Appendix E.3. Together, these results show that greedy hierarchical reuse is near-optimal on tractable instances and that explicit trajectory control still preserves substantial reuse.

Table 1: Small-instance token savings against the unconstrained DP optimum.

| Mixture family | Unconstrained DP ↑ | `MTR-a` ↑ | `MTR-r` ↑ | `Star` ↑ |
|---|---|---|---|---|
| Sparse Dirichlet | 42.27% | **42.20%** | 35.80% | 0.00% |
| Uniform Dirichlet | 67.22% | **66.95%** | 50.92% | 45.37% |
| RegMix-like | 38.53% | **38.13%** | 21.60% | 2.62% |
| Overall mean | 49.34% | **49.09%** | 36.10% | 16.00% |

**Scaling with mixtures and domains.** Tree reuse becomes more valuable as the number of targets grows (Table 2). Averaged over eleven mixture families, token savings increase from $65.20\%$ to $84.60\%$ for `MTR-a` and from $45.80\%$ to $72.10\%$ for `MTR-r` as $m$ grows from 50 to 2000. By contrast, `Star` decreases from $8.70\%$ to $3.60\%$ over the same range. On the other hand, token savings decrease as the domain dimension grows, yet both tree-based methods retain substantial reuse even at $d = 128$, where `MTR-a` and `MTR-r` save $36.00\%$ and $20.10\%$ of tokens. The complete Forward-TV threshold sweep and per-family scaling results are reported in Appendices E.5 and E.6.

Table 2: Overall mean token savings versus $m$ at $d = 8$ and versus $d$ at $m = 300$.

| $m$-scaling at $d = 8$ | | | | $d$-scaling at $m = 300$ | | | |
|---|---|---|---|---|---|---|---|
| $m$ | `MTR-a` | `MTR-r` | `Star` | $d$ | `MTR-a` | `MTR-r` | `Star` |
| 50 | **65.20%** | 45.80% | 8.70% | 4 | **90.40%** | 84.70% | 7.10% |
| 100 | **71.00%** | 52.20% | 7.10% | 8 | **77.30%** | 60.20% | 5.60% |
| 200 | **75.20%** | 57.10% | 5.90% | 10 | **73.50%** | 53.90% | 4.80% |
| 300 | **77.30%** | 60.20% | 5.60% | 16 | **65.00%** | 41.30% | 3.00% |
| 500 | **79.70%** | 63.80% | 4.90% | 32 | **53.70%** | 30.60% | 2.00% |
| 1000 | **82.40%** | 68.10% | 4.20% | 64 | **44.10%** | 24.50% | 2.00% |
| 2000 | **84.60%** | 72.10% | 3.60% | 128 | **36.00%** | 20.10% | 2.00% |

### 7.2 REAL-WORLD EXPERIMENTS ON ESSENTIAL-WEB

This study examines whether the structural reuse identified in the synthetic experiments translates into end-to-end proxy-training savings while preserving the RegMix fit used for mixture selection. We first establish the real-world training setting, then evaluate MTR's computational efficiency and compositional fidelity. This progression connects the reuse topology produced by the planner to both realized GPU savings and the quality of the fitted RegMix surrogate.

**Dataset and training environment.** We construct 128 10-domain proxy-training mixtures from Essential-Web v1.0 (Essential AI et al., 2025), with approximately 1B training tokens allocated to

each proxy. Each proxy is a 0.98M-parameter decoder-only model trained on NVIDIA RTX 4090 GPUs. The complete data provenance and optimization configuration are provided in Appendix E.8. All results are averaged over six paired random seeds, with model initialization, sampling randomness, and validation controlled within each seed as detailed in Appendix E.1. Note that `Star` fails to train an effective shared prefix in this real-world experiment and is equivalent to rerunning the no-reuse baseline, as the coordinatewise common prefix of all 128 mixtures is zero.

**Computational efficiency.** Hierarchical reuse converts structural overlap into substantial end-to-end training savings. As shown in Table 3, `MTR-a` avoids 70.0% of the tokens and completes in 7.92 aggregate GPU-hours. With Forward-TV control, `MTR-r` retains 49.0% token savings and completes in 13.91 GPU-hours, a 48.3% reduction from the 26.93 GPU-hours required by `Strict`.

Table 3: Token savings and RegMix fidelity on 128 Essential-Web proxy mixtures over six paired random seeds (mean ± standard deviation, details in Appendix E.2). $L_1$ and maximum domain deviation compare the mean top-128 candidate composition with `Strict`, with the deviation reported in percentage points. Proxy Spearman compares the ordering of top-128 observed proxy losses.

| Method | Saving ↑ | GPU-h ↓ | $L_1$ ↓ | Max deviation ↓ | Proxy Spearman ↑ |
|---|---|---|---|---|---|
| `Strict` | 0.0% | $26.93 \pm 0.10$ | **0** | **0** | **1** |
| `Star` | 0.0% | $26.93 \pm 0.10$ | **0** | **0** | **1** |
| `MTR-a` | **70.0%** | $\mathbf{7.92 \pm 0.21}$ | $.2543 \pm .0500$ | $10.29 \pm 2.22$ | $.8481 \pm .0172$ |
| `MTR-r` | 49.0% | $13.91 \pm 0.14$ | $\mathbf{.1615 \pm .0483}$ | $\mathbf{6.52 \pm 2.89}$ | $\mathbf{.9516 \pm .0140}$ |

**Compositional fidelity.** Forward-TV converts the unconstrained savings of `MTR-a` into a substantially closer match to independent RegMix fitting. In Table 3, `MTR-r` reduces the top-128 composition $L_1$ distance from 0.2543 to 0.1615 and the maximum domain deviation from 10.29 to 6.52 percentage points relative to `MTR-a`. It also raises Proxy Spearman from 0.8481 to 0.9516. Thus, `MTR-r` preserves the selected composition and proxy ordering while saving nearly half of the training tokens.

**Summary.** The Essential-Web study establishes a clear efficiency–fidelity progression across the three executed strategies. `MTR-a` exposes the maximum savings available from unconstrained hierarchical reuse, while `MTR-r` retains 49.0% token savings and reduces aggregate GPU time by 48.3% with substantially stronger top-128 compositional fidelity and proxy-loss alignment. These results show that Forward-TV retains substantial savings from hierarchical checkpoint reuse while preserving the RegMix selection outcome of independent training.

## 8 Conclusion

We introduced MTR, a framework for reducing the cost of RegMix proxy swarms by organizing overlapping target mixtures into checkpoint-reuse trees. `MTR-a` provides a provable greedy planner for arbitrary-order reuse, while `MTR-r` uses Forward-TV constraints to balance training efficiency and trajectory fidelity. Experiments on synthetic mixtures and Essential-Web show that MTR substantially reduces token consumption and aggregate GPU time while preserving RegMix selection quality.

### AI use statement

In this work, we used generative AI tools for feedback on the research methodology, code editing and debugging, and drafting and formatting the manuscript. We have not used generative AI tools to generate or alter experimental measurements, and the remaining required-disclosure tasks, such as human-subjects research and the generation of image, audio, or video content presented as evidence, are not applicable to this work. We have reviewed all AI-assisted work: the authors reviewed and tested the AI-assisted code, and checked the reported numerical results against the recorded experiment artifacts. We take responsibility for the final content of this work, including text, claims, or artifacts produced with the aid of generative AI.

### Reproducibility statement

Code and configuration files are available at `https://anonymous.4open.science/r/mtr-45CD/`. Essential-Web can be obtained from its official source under its stated license; the repository provides the preprocessing and evaluation scripts needed to reproduce the reported results.

## References


Mayee F. Chen, Michael Y. Hu, Nicholas Lourie, Kyunghyun Cho, and Christopher Ré. AIOLI: A unified optimization framework for language model data mixing. In *International Conference on Learning Representations*, 2025.

Mayee F. Chen, Tyler Murray, David Heineman, Matt Jordan, Hannaneh Hajishirzi, Christopher Ré, Luca Soldaini, and Kyle Lo. Olmix: A framework for data mixing throughout LM development, 2026a. *arXiv preprint arXiv:2602.12237*.

Zhuo Chen, Yuxuan Miao, and Deyi Xiong. Data mixing for large language models pretraining: A survey and outlook. *Data Intelligence*, 8(1):15–55, 2026b. doi: 10.3724/2096-7004.di.2026.0055.

Shizhe Diao, Yu Yang, Yonggan Fu, Xin Dong, Dan Su, Markus Kliegl, Zijia Chen, Peter Belcak, Yoshi Suhara, Hongxu Yin, Mostofa Patwary, Yingyan Lin, Jan Kautz, and Pavlo Molchanov. Nemotron-CLIMB: Clustering-based iterative data mixture bootstrapping for language model pre-training. In *Advances in Neural Information Processing Systems*, volume 38, 2025. doi: 10.52202/085713-0865.

Dorit Dor and Michael Tarsi. Graph decomposition is $NP$-complete: A complete proof of Holyer's conjecture. *SIAM Journal on Computing*, 26(4):1166–1187, 1997. doi: 10.1137/S0097539792241237.

Essential AI, Andrew Hojel, Michael Pust, Tim Romanski, Yash Vanjani, Ritvik Kapila, Mohit Parmar, Adarsh Chaluvaraju, Alok Tripathy, Anil Thomas, Ashish Tanwer, Darsh J. Shah, Ishaan Shah, Karl Stratos, Khoi Nguyen, Kurt Smith, Michael Callahan, Peter Rushton, Philip Monk, Platon Mazarakis, Saad Jamal, Saurabh Srivastava, Somanshu Singla, and Ashish Vaswani. Essential-Web v1.0: 24T tokens of organized web data, 2025. *arXiv preprint arXiv:2506.14111*.

Simin Fan, Matteo Pagliardini, and Martin Jaggi. DoGE: Domain reweighting with generalization estimation. In *Proceedings of the 41st International Conference on Machine Learning*, volume 235, pp. 12895–12915, 2024.

Xinran Gu, Kaifeng Lyu, Jiazheng Li, and Jingzhao Zhang. Data mixing can induce phase transitions in knowledge acquisition. In *Advances in Neural Information Processing Systems*, volume 38, pp. 181110–181154, 2025. doi: 10.52202/085713-5449.

Michael Y. Hu, Apurva Gandhi, Kyunghyun Cho, Tal Linzen, and Pratyusha Sharma. Always learning, always mixing: Efficient and simple data mixing all the time, 2026. *arXiv preprint arXiv:2605.15220*.

Yiding Jiang, Allan Zhou, Zhili Feng, Sadhika Malladi, and J. Zico Kolter. Adaptive data optimization: Dynamic sample selection with scaling laws. In *International Conference on Learning Representations*, 2025.

Alaa Khaddaj, Logan Engstrom, and Aleksander Madry. Small-to-large generalization: Data influences models consistently across scale. In *International Conference on Learning Representations*, 2025.

Shengrui Li, Fei Zhao, Kaiyan Zhao, Jieying Ye, Haifeng Liu, Fangcheng Shi, Zheyong Xie, Yao Hu, and Shaosheng Cao. Decouple searching from training: Scaling data mixing via model merging for large language model pre-training. In *International Conference on Machine Learning*, 2026.

Qian Liu, Xiaosen Zheng, Niklas Muennighoff, Guangtao Zeng, Longxu Dou, Tianyu Pang, Jing Jiang, and Min Lin. RegMix: Data mixture as regression for language model pre-training. In *International Conference on Learning Representations*, 2025.

Yuri A. Malkov and Dmitry A. Yashunin. Efficient and robust approximate nearest neighbor search using hierarchical navigable small world graphs. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 42(4):824–836, 2020. doi: 10.1109/TPAMI.2018.2889473.

Microsoft AI Team et al. Mai-thinking-1: Building a hill-climbing machine. Technical report, Technical report, Microsoft AI, 2026., 2025.

Jiachen T. Wang, Tong Wu, Kaifeng Lyu, James Zou, Dawn Song, Ruoxi Jia, and Prateek Mittal. Can small training runs reliably guide data curation? Rethinking proxy-model practice. In *International Conference on Learning Representations*, 2026a.

Jiapeng Wang, Changxin Tian, Kunlong Chen, Ziqi Liu, Jiaxin Mao, Wayne Xin Zhao, Zhiqiang Zhang, and Jun Zhou. MergeMix: Optimizing mid-training data mixtures via learnable model merging, 2026b. *arXiv preprint arXiv:2601.17858*.

Alexander Wettig, Kyle Lo, Sewon Min, Hannaneh Hajishirzi, Danqi Chen, and Luca Soldaini. Organize the web: Constructing domains enhances pre-training data curation. In *Proceedings of the 42nd International Conference on Machine Learning*, volume 267, pp. 66678–66705, 2025.

Sang Michael Xie, Hieu Pham, Xuanyi Dong, Nan Du, Hanxiao Liu, Yifeng Lu, Percy Liang, Quoc V. Le, Tengyu Ma, and Adams Wei Yu. DoReMi: Optimizing data mixtures speeds up language model pretraining. In *Advances in Neural Information Processing Systems*, volume 36, pp. 69798–69818, 2023. doi: 10.52202/075280-3059.

Jiasheng Ye, Peiju Liu, Tianxiang Sun, Jun Zhan, Yunhua Zhou, and Xipeng Qiu. Data mixing laws: Optimizing data mixtures by predicting language modeling performance. In *International Conference on Learning Representations*, 2025.

Kevin Zhou, Lisa Alazraki, Kris Cao, and Marek Rei. Repetition mismatch: Why data mixture experiments don't scale and how to fix them. In *Conference on Empirical Methods in Natural Language Processing*, 2026.

# A PROOFS

**Lemma 1** (Best checkpoint for a descendant set)**.** *Fix a feasible tree topology. For every internal node $u$ with descendant leaf set $S(u)$, the cost is minimized by the checkpoint $c(u) = q(S(u))$.*

*Proof of Lemma 1.* Any checkpoint $c(u)$ that can serve all descendant leaves must satisfy

$$c(u) \le p^{(j)} \qquad \text{for every } j \in S(u).$$

Hence, $c(u) \le q(S(u))$ coordinatewise. We now show that this upper bound is feasible and optimal.

First, the assignment $c(u) = q(S(u))$ is feasible. For any edge $(u, v)$ with $u$ the parent of $v$, the set $S(v)$ of descendant leaves of $v$ is contained in $S(u)$, so

$$c(u) = \min_{j \in S(u)} p^{(j)} \le \min_{j \in S(v)} p^{(j)} = c(v),$$

where for a leaf $v$ the set $S(v)$ is the singleton $\{v\}$ and $c(v) = p^{(v)}$. Since also $c(u) \ge 0$, every edge is monotone and the assignment is feasible.

Second, the assignment $c(u) = q(S(u))$ is optimal. Because every edge is monotone, $\|c(v) - c(u)\|_1 = \|c(v)\|_1 - \|c(u)\|_1$ for each edge $(u, v)$, so summing over the edges counts each non-root internal node $u$ once as a child and $\mathrm{ch}(u)$ times as a parent, where $\mathrm{ch}(u)$ is the number of children of $u$. Hence

$$\mathrm{cost}(T) = m - \sum_u \big(\mathrm{ch}(u) - 1\big) \, \|c(u)\|_1 \,,$$

where the sum ranges over the non-root internal nodes of $T$. Since $\mathrm{ch}(u) \ge 1$ and $c(u) \le q(S(u))$ imply $\|c(u)\|_1 \le \|q(S(u))\|_1$, every feasible assignment satisfies

$$\mathrm{cost}(T) \ge m - \sum_u \big(\mathrm{ch}(u) - 1\big) \, \|q(S(u))\|_1 \,,$$

where the assignment $c(u) = q(S(u))$ attains this lower bound. Therefore the cost is minimized by $c(u) = q(S(u))$. □

**Theorem 1** (Equivalence of MTR-a and laminar tree selection)**.** *Maximizing $\Phi(\mathcal{L})$ over laminar families $\mathcal{L}$ containing $[m]$ and every singleton is equivalent to MTR-a: every checkpoint-relabeled MTR-a tree $T$ admits such a family with $\mathrm{cost}(T) = m - \Phi(\mathcal{L})$, and conversely every such family $\mathcal{L}$ is realized by a feasible MTR-a tree $T$ with $\mathrm{cost}(T) = m - \Phi(\mathcal{L})$. Consequently, $\min_T \mathrm{cost}(T) = m - \max_{\mathcal{L}} \Phi(\mathcal{L})$, so the two problems are equivalent.*

*Proof of Theorem 1.* Let $T$ be a feasible MTR-a tree, and let $T'$ be obtained from $T$ by deleting every internal node with a single child. Deleting these nodes preserves the cost, so $\mathrm{cost}(T) = \mathrm{cost}(T')$.

Consider the family of descendant sets of $T'$:

$$\mathcal{L}(T') = \{S(u) : u \in V(T')\} \,.$$

Every internal node of $T'$ has at least two children, so two distinct nodes of $T'$ have distinct descendant sets, and $\mathcal{L}(T')$ contains $[m]$ from the root and the singletons from the leaves. Descendant sets are nested along any root-to-leaf path and disjoint across different branches, so $\mathcal{L}(T')$ is laminar, and summing the optimal contribution of every internal node in Lemma 1 gives $\mathrm{cost}(T') = m - \Phi(\mathcal{L}(T'))$.

Conversely, let $\mathcal{L}$ contain $[m]$ and every singleton and be laminar, and let $T_{\mathcal{L}}$ be the Hasse diagram of $\mathcal{L}$, with the root $[m]$ labeled by 0 and every other set $S$ labeled by $q(S)$. Every child set is a proper subset of its parent set, so Lemma 1 shows that each edge is monotone and that these labels are optimal. For every non-singleton set $S \in \mathcal{L}$, the maximal proper subsets of $S$ are pairwise disjoint and cover $S$, so every non-singleton set has at least two children and hence $\mathcal{L}(T_{\mathcal{L}}) = \mathcal{L}$. The same cost identity therefore gives $\mathrm{cost}(T_{\mathcal{L}}) = m - \Phi(\mathcal{L})$.

The two constructions are inverse to each other on checkpoint-relabeled trees without unary internal nodes, so they form a bijection between such trees and laminar families that contain $[m]$ and every singleton. Minimizing the cost is therefore equivalent to maximizing the reuse gain, which proves the claimed equality of optimal values. □

**Theorem 2** (NP-hardness of MTR-a)**.** *The decision version of MTR-a is NP-complete.*

*Proof of Theorem 2.* We reduce from *triangle decomposition*: given a simple undirected graph $G = (V, E)$ with $|E| = 3r$, decide whether $E$ can be partitioned into $r$ edge-disjoint triangles. This problem is NP-complete (Dor & Tarsi, 1997). Let

$$F = \{A \subseteq E : A \text{ is the three-edge set of a triangle in } G\}.$$

We build an MTR-a instance with one target per edge $e \in E$, so $m = |E| = 3r$. Create one coordinate for each $A \in F$ and one private coordinate for each $e \in E$, giving dimension $d = |F| + |E|$. Define

$$\deg_F(e) = \big|\{A \in F : e \in A\}\big|, \qquad \alpha = \frac{1}{|F| + 1}.$$

For each $e \in E$, define a target vector $p^{(e)} \in \mathbb{R}^d_{\geq 0}$ by

$$p^{(e)}_A = \begin{cases} \alpha, & e \in A, \\ 0, & e \notin A, \end{cases} \qquad p^{(e)}_e = 1 - \deg_F(e)\alpha, \qquad p^{(e)}_i = 0 \text{ for } i \in E \setminus \{e\}.$$

Since $\deg_F(e)\alpha \leq |F|\alpha < 1$, each $p^{(e)}$ is non-negative and satisfies $\big\|p^{(e)}\big\|_1 = 1$. The construction is polynomial time by enumerating all triangles in $G$ (at most $|V|^3$) and writing down the resulting rational targets.

Since the leaves are indexed by edges, every internal node $u$ of a feasible tree is naturally associated with its descendant edge set $S(u) \subseteq E$. For every feasible tree $T$, by Lemma 1, relabeling every internal node $u$ by $q(S(u))$ is feasible and cannot increase the cost, and deleting an internal node with a single child preserves both the cost and the descendant sets of the remaining nodes. Hence we may always assume that a feasible tree is checkpoint-relabeled and has no unary internal nodes hereafter, in which case Theorem 1 gives

$$\mathrm{cost}(T) = m - \Phi(T) = 3r - \Phi(T),$$

where $\Phi(T) = \sum_{u \in U(T)} \big(\mathrm{ch}_T(u) - 1\big) w\big(S(u)\big)$. Let $b = 3r - 2r\alpha$. Lemma 3 below shows that $\Phi(T) \leq 2r\alpha$; furthermore, if $\Phi(T) = 2r\alpha$, then $E$ admits a triangle decomposition. Indeed, if $G$ has a triangle decomposition

$$E = A_1 \sqcup \cdots \sqcup A_r, \text{ such that } A_t \in F \text{ for all } t \in [r],$$

then making each $A_t$ the leaf set of a local subtree and joining those $r$ subtrees above them yields $\Phi(T) = 2r\alpha$ and hence $\mathrm{cost}(T) = b$. Conversely, if some feasible tree satisfies $\mathrm{cost}(T) \leq b$, then the inequality above gives $\Phi(T) \geq m - \mathrm{cost}(T) \geq 2r\alpha$, which implies $\Phi(T) = 2r\alpha$ and a triangle decomposition of $E$ by Lemma 3. Therefore $G$ has a triangle decomposition if and only if the constructed instance has optimum cost at most $b$. Membership in NP is immediate, so the decision version of MTR-a is NP-complete.

**Lemma 3** (An upper bound on total overlap)**.** *Let $T$ be any feasible MTR-a tree for the constructed instance, with $\Phi(T)$ defined as above. Then, $\Phi(T) \leq 2r\alpha$. Moreover, if $\Phi(T) = 2r\alpha$, then $E$ admits a triangle decomposition.*

*Proof.* We first record the possible overlap weights. For every $S \subseteq E$ with $|S| \geq 2$, no private coordinate contributes to $q(S)$, and a triangle coordinate $A \in F$ contributes $\alpha$ if and only if $S \subseteq A$. Since $G$ is simple, any two distinct edges lie in at most one triangle: disjoint edges lie in none, and two edges sharing a vertex determine the third edge uniquely. Hence

$$w(S) \in \{0, \alpha\}, \qquad w(S) = \alpha \iff S \subseteq A \text{ for some } A \in F.$$

Call an internal node *positive* if $w(S(u)) = \alpha$, and let $\mathcal{M} = \{M_1, \ldots, M_s\}$ be an arbitrary collection of disjoint descendant sets, each derived from a positive node. These disjoint descendant sets of internal nodes in a rooted tree form a laminar family. Since $w(S) = \alpha$ implies $S \subseteq A$ for some $A \in F$ and each such $A$ has size 3, every $M_t$ has size 2 or 3. Let $r_3$ be the number of sets of size 3 and let $r_2$ be the number of sets of size 2. Disjointness gives

$$3r_3 + 2r_2 \leq |E| = 3r.$$

Hence $r_3 \le r$. For a positive node with leaf set of size $t$, every internal node in its subtree with at least two children has weight $\alpha$, and any rooted tree on $t$ leaves satisfies

$$\sum_u \big(\mathrm{ch}_T(u) - 1\big) = t - 1$$

over the internal nodes in that subtree. Thus the total contribution from that subtree is at most $(t-1)\alpha$, so

$$\Phi(T) \le (2r_3 + r_2)\alpha.$$

Finally,

$$2r_3 + r_2 = \frac{(3r_3 + 2r_2) + r_3}{2} \le \frac{3r + r}{2} = 2r,$$

so $\Phi(T) \le 2r\alpha$.

If $\Phi(T) = 2r\alpha$, then $2r_3 + r_2 = 2r$, which forces $r_3 = r$ and $r_2 = 0$. Thus the disjoint positive sets are $r$ disjoint triples whose union has size $3r = |E|$, so they partition $E$. Therefore these $r$ triples form a triangle decomposition of $E$. □

□

**Lemma 4.** *The greedy merge weights are nonincreasing, meaning $g_1 \ge g_2 \ge \cdots \ge g_{m-1}$.*

*Proof of Lemma 4.* Fix one iteration, and suppose the greedy rule merges the current blocks $A$ and $B$. Write $S = A \cup B$, so the merge chosen at this iteration has weight $g = w(S)$. Consider any candidate merge that is available immediately after replacing $A$ and $B$ by $S$. There are two possibilities. If the candidate merge joins two blocks different from $S$, then it was already available before the current iteration, so its weight is at most $g$ by the greedy choice of $A$ and $B$. If the candidate merge is of the form $S \cup C$ for some other current block $C$, then before the merge both $A \cup C$ and $B \cup C$ were feasible candidates. Since $A \cup C \subseteq S \cup C$ and $B \cup C \subseteq S \cup C$, the monotonicity Lemma 2 gives

$$w(S \cup C) \le w(A \cup C) \qquad \text{and} \qquad w(S \cup C) \le w(B \cup C).$$

Both $A \cup C$ and $B \cup C$ had weight at most $g$ before the merge, so $w(S \cup C) \le g$ as well. Thus every merge available at the next iteration has weight at most $g$, and therefore the next greedy merge weight is at most the current one. Repeating this argument at every iteration proves that $g_1 \ge g_2 \ge \cdots \ge g_{m-1}$. □

**Lemma 5.** *For any threshold $\tau \in [0, 1]$, we have*

$$n_{T^\star}(\tau) \le 2n_{T_{\mathrm{g}}}(\tau).$$

*Proof of Lemma 5.* Fix a threshold $\tau \in [0, 1]$, we will derive the inequality by comparing two partitions from $T_{\mathrm{g}}$ and $T^\star$. First, consider the greedy tree $T_{\mathrm{g}}$. Let $t = n_{T_{\mathrm{g}}}(\tau)$. Since $\Phi(T_{\mathrm{g}}) = \sum_{s=1}^{m-1} g_s$ and $g_1 \ge \cdots \ge g_{m-1}$ by Lemma 4, the number $t$ is exactly the number of greedy merge steps whose weight is strictly larger than $\tau$. Let $\mathcal{P}$ denote this greedy partition after the first $t$ merges.

Now delete from $T^\star$ every node $u \in U(T^\star)$ with $w(u) \le \tau$ while keeping all leaves. By monotonicity of $w$ under inclusion (Lemma 2), one has $w(\mathrm{parent}(u)) \le w(u)$ for every internal edge, so the remaining nodes with $w(u) > \tau$ form a forest of full binary subtrees. Let $\mathcal{S} = \{S_1, \ldots, S_k\}$ be the leaf sets of the connected components of this forest, and let $r_a = |S_a|$ for each $a \in [k]$. Each component is a full binary tree on $r_a$ leaves, so it contributes exactly $r_a - 1$ nodes to $n_{T^\star}(\tau)$. Therefore one has the identity $n_{T^\star}(\tau) = \sum_{a=1}^{k}(r_a - 1) = \sum_{a=1}^{k} r_a - k$.

Next, let us compare the two partitions $\mathcal{P}$ and $\mathcal{S}$. For each $a \in [k]$, let $s_a$ be the number of blocks $C \in \mathcal{P}$ satisfying $C \subseteq S_a$. Initially every leaf is a singleton block, so one has $\sum_{a=1}^{k} s_a = \sum_{a=1}^{k} r_a$ before any merges occur. Each greedy merge removes two blocks and replaces them by their union, so in one merge the quantity $\sum_{a=1}^{k} s_a$ can decrease by at most two. After $t$ merges this implies that $\sum_{a=1}^{k} s_a \ge \sum_{a=1}^{k} r_a - 2t$. On the other hand, one must have $s_a \le 1$ for every $a \in [k]$. Indeed, if some $a$ had two distinct blocks $C, D \in \mathcal{P}$ with $C \subseteq S_a$ and $D \subseteq S_a$, then $C \cup D \subseteq S_a$ and the

monotonicity Lemma 2 would imply $w(C \cup D) \geq w(S_a) > \tau$, which contradicts the fact that $\mathcal{P}$ admits no merge of weight larger than $\tau$. Therefore $\sum_{a=1}^{k} s_a \leq k$, and combining the two bounds yields $\sum_{a=1}^{k} r_a - k \leq 2t$. Using the identity for $n_{T^\star}(\tau)$, this shows that $n_{T^\star}(\tau) \leq 2n_{T_{\mathrm{g}}}(\tau)$. □

**Theorem 3.** *Let $T_{\mathrm{g}}$ be the greedy agglomeration tree defined above, and let $T^\star$ be an optimal MTR-a tree. Then $\Phi(T_{\mathrm{g}}) \geq \frac{1}{2}\Phi(T^\star)$, and equivalently $\mathrm{cost}(T_{\mathrm{g}}) \leq \frac{1}{2}\big(m + \mathrm{cost}(T^\star)\big)$.*

*Proof of Theorem 3.* By Lemma 6 below, we can assume without loss of generality that $T^\star$ is binary. For any feasible binary tree $T$, write $w(u) = w\big(S(u)\big)$ for each $u \in U(T)$, so that $\Phi(T) = \sum_{u \in U(T)} w(u)$. For every threshold $\tau \in [0, 1]$, define $n_T(\tau) = \big|\{u \in U(T) : w(u) > \tau\}\big|$. Since each $w(u) \in [0, 1]$, the layer-cake identity yields

$$\Phi(T) = \sum_{u \in U(T)} w(u) = \int_0^1 n_T(\tau)\, d\tau.$$

Thus, our main idea is to compare $n_{T_{\mathrm{g}}}(\tau)$ and $n_{T^\star}(\tau)$ for every $\tau \in [0, 1]$ to establish a bound between $\Phi(T_{\mathrm{g}})$ and $\Phi(T^\star)$. We first discuss the case of an arbitrary fixed threshold $\tau \in [0, 1]$, and we will show in Lemma 5 it holds that $n_{T^\star}(\tau) \leq 2n_{T_{\mathrm{g}}}(\tau)$. Therefore, integrating this inequality over $\tau$ and applying the layer-cake identity yields $\Phi(T^\star) \leq 2\Phi(T_{\mathrm{g}})$.

**Lemma 6.** *There exists an optimal MTR-a tree $T^\star$ maximizing $\Phi(T)$ such that every node in $U(T^\star)$ has exactly two children.*

*Proof.* Let $T$ be any feasible tree, and assume without loss of generality that $T$ is checkpoint-relabeled by Lemma 1. Let $u \in U(T)$ have $k \geq 3$ children whose descendant leaf sets are $S_1, \ldots, S_k$. Let $S = \bigcup_{t=1}^{k} S_t$ be the descendant leaf set of $u$. Replace the local star formed by $u$ and its $k$ child subtrees by any binary tree whose leaves are exactly these $k$ subtrees, and label each internal node $v$ in this binary refinement by $q\big(S(v)\big)$. This binary refinement has exactly $k - 1$ internal nodes. Moreover, every such internal node satisfies $\varnothing \neq S(v) \subseteq S$, so the monotonicity Lemma 2 gives $w\big(S(v)\big) \geq w(S)$. The original node $u$ contributes $(k-1)w(S)$ to $\Phi(T)$, while the refined binary gadget contributes

$$\sum_{v} w\big(S(v)\big) \geq (k-1)w(S),$$

where the sum runs over the $k - 1$ internal nodes of the refinement. Therefore this replacement does not decrease $\Phi(T)$. Iterating this operation yields a feasible binary tree with reuse gain at least that of $T$, so some optimal maximizer $T^\star$ can be chosen binary on $U(T^\star)$. □

□

**Lemma 2.** *If $A \subseteq B$ are nonempty sets, then $w(A) \geq w(B)$.*

*Proof of Lemma 2.* The definition of $q(S)$ gives $q(B) \leq q(A)$ coordinatewise when $A \subseteq B$. Therefore $\|q(B)\|_1 \leq \|q(A)\|_1$, which is exactly $w(B) \leq w(A)$. □

**Theorem 4.** *For every $\varepsilon > 0$, there exists an MTR-a instance for which the greedy agglomeration tree $T_{\mathrm{g}}$ satisfies*

$$\Phi(T_{\mathrm{g}}) \leq \big(\tfrac{1}{2} + \varepsilon\big)\Phi(T^\star),$$

*where $T^\star$ is an optimal MTR-a tree.*

*Proof of Theorem 4.* Fix any $x \in (1/3, 1/2)$, and consider four targets in dimension $d = 5$ given by

$$p^{(1)} = (1{-}x, 0, x, 0, 0),\ p^{(2)} = (0, 1{-}x, x, 0, 0),\ p^{(3)} = (1{-}x, 0, 0, x, 0),\ p^{(4)} = (0, 0, 0, x, 1{-}x).$$

For the pairwise overlap weights, a direct calculation gives

$$w(\{1,3\}) = 1{-}x, \qquad w(\{1,2\}) = w(\{3,4\}) = x, \qquad w(\{1,4\}) = w(\{2,3\}) = w(\{2,4\}) = 0.$$

Since $x < 1/2$, the unique maximum pairwise overlap is $w(\{1,3\}) = 1-x$, so the greedy rule must merge $\{1\}$ and $\{3\}$ first. Let $S = \{1,3\}$. One then has

$$w(S \cup \{2\}) = w(\{1,2,3\}) = 0, \qquad w(S \cup \{4\}) = w(\{1,3,4\}) = 0, \qquad w(\{2,4\}) = 0,$$

and therefore every subsequent greedy merge has overlap weight zero. It follows that $\Phi(T_{\mathrm{g}}) = 1-x$.

On the other hand, the binary tree that first merges $\{1,2\}$ and $\{3,4\}$ and then merges their parents has reuse gain

$$\Phi(T) = x + x + 0 = 2x.$$

Hence $\Phi(T^\star) \geq 2x$, and therefore

$$\frac{\Phi(T_{\mathrm{g}})}{\Phi(T^\star)} \leq \frac{1-x}{2x}.$$

As $x \to 1/2$, the right-hand side tends to $1/2$. Thus, for every $\varepsilon > 0$, choosing $x$ sufficiently close to $1/2$ yields

$$\Phi(T_{\mathrm{g}}) \leq \left(\tfrac{1}{2} + \varepsilon\right)\Phi(T^\star),$$

which shows that the factor $1/2$ in Theorem 3 is tight for this greedy merge rule. □

## B Expanded Related Work

**Offline data-mixture search.** Offline methods use auxiliary training runs to select a fixed mixture before the principal training run, a broad family surveyed by Chen et al. (2026b). DoReMi trains a proxy model with group distributionally robust optimization and transfers the resulting domain weights to a larger model (Xie et al., 2023). RegMix instead samples many mixtures, trains a proxy model for each mixture, and fits a regression model that predicts the performance of unseen mixtures (Liu et al., 2025). *Olmix* systematizes this surrogate-based workflow and introduces *mixture reuse*, which retains ratios from an existing mixture when the domain set changes (Chen et al., 2026a). When domains are not given in advance, CLIMB alternates between clustering an unlabeled corpus and searching for mixtures over the induced domains (Diao et al., 2025), while WebOrganizer constructs explicit topic and format domains before mixture optimization (Wettig et al., 2025). MTR is complementary to all of these methods because it optimizes how the corresponding training runs share checkpoints among candidate mixtures.

**Online mixture reweighting.** Online methods adjust domain weights during training instead of committing to a mixture selected by a separate offline search. Adaptive Data Optimization fits per-domain scaling laws during training and allocates tokens according to the estimated learning potential of each domain, without training an external proxy model (Jiang et al., 2025). AIOLI estimates the parameters of its mixing law online, and updates the mixture throughout training (Chen et al., 2025). These methods optimize a single evolving data trajectory rather than a family of independently specified candidate mixtures, albeit at the cost of higher training variance.

**Domain expert merging.** Another line of work reduces mixture-search cost by training domain-specific parameter components and combining them to approximate candidate mixtures. DeMix (Li et al., 2026) and MergeMix (Wang et al., 2026b) train one component model per candidate dataset and evaluate arbitrary mixtures through weighted model merging. OP-Mix operates on the current model by training low-rank adapters on individual domains and interpolating the adapters to simulate candidate mixtures (Hu et al., 2026). However, domain merging may fail to accurately capture complex interaction between domains.

**Reliability across scales and horizons.** The use of inexpensive mixture experiments depends on whether their conclusions transfer across model scales, optimization configurations, and training horizons. Data Mixing Laws fits an explicit parametric law that predicts language modeling performance from the mixture proportions, which permits extrapolation to mixtures and token budgets beyond those observed in the proxy runs (Ye et al., 2025). Small-to-large analyses find that the effect of data on model predictions is often correlated across scale, but also document systematic differences between small and large models (Khaddaj et al., 2025). Proxy-model rankings can also reverse under data-dependent hyperparameter choices, which motivates evaluating each recipe under an appropriate training configuration (Wang et al., 2026a). Mixture transfer becomes still less

predictable when knowledge acquisition exhibits ratio- and scale-dependent phase transitions (Gu et al., 2025) or when small and large runs expose scarce domains to different repetition rates (Zhou et al., 2026). These are orthogonal to the planning problem MTR addresses.

# C FAST AGGLOMERATIVE IMPLEMENTATION

The direct greedy implementation requires $O(m^3 d)$ time, while an optimized exact implementation reduces this complexity to $O(m^2(d + \log m))$ at the cost of $O(m^2)$ memory. We review both exact implementations and then propose an approximately greedy variant that scales to large $m$ by restricting merge proposals to approximate nearest neighbors of the current block checkpoints.

We first inspect the time complexity of the exact agglomerative algorithm. Since $w(A \cup B) = \|q(A \cup B)\|_1$, evaluating one candidate merge takes $O(d)$ time. In a direct implementation, when $r$ blocks remain one has to evaluate all $\binom{r}{2}$ candidate merges in $O(r^2 d)$ time, which totals $O(m^3 d)$ over $r = m, m-1, \ldots, 2$. An exact implementation can be improved to $O(m^2(d+\log m))$ time and $O(m^2)$ memory by storing all pairwise merge scores in a max-priority queue and lazily updating only pairs involving the newly formed block after each merge. When $r$ blocks remain, such a merge rescores the $r - 2$ new pairs in $O(d)$ time each and performs $O(\log m)$ queue work per pair, so one round costs $O(r(d + \log m))$ and all rounds cost $O(m^2(d + \log m))$.

When $m$ is very large, the $O(m^2)$ footprint is prohibitive. We propose using an approximately greedy sparse search. One practical option is to maintain a graph-based approximate nearest-neighbor (ANN) index over the current block checkpoints and to restrict merge proposals to approximate neighbors. We adopt the hierarchical navigable small world (HNSW) index (Malkov & Yashunin, 2020). An ANN query requires a similarity metric, and we use the $\ell_1$ distance $\|q(A) - q(B)\|_1$ as a proxy for the true merge objective. This choice is motivated by the identity

$$w(A \cup B) = \|\min\{q(A), q(B)\}\|_1 = \tfrac{1}{2}\Big(\|q(A)\|_1 + \|q(B)\|_1 - \|q(A) - q(B)\|_1\Big),$$

which holds because $q(A)$ and $q(B)$ are nonnegative checkpoints. Because the individual weights vary across blocks, we use the distance only to generate merge proposals.

For each block $A$ we query HNSW for a small candidate set $N(A)$ of approximate $\ell_1$-nearest neighbors, and we define $B(A)$ as the element of $N(A)$ maximizing the exact score $w(A \cup B)$. We compute the exact score for each proposed pair $(A, B(A))$ and choose the merge with the largest cached score over all current blocks. To avoid re-querying all blocks after each merge, we cache for each block $A$ its latest proposed partner $B(A)$ and its exact score $w(A \cup B(A))$, which is stored in a max-priority queue. We refresh a block's ANN query only when it is popped from the queue and its cached partner has been merged elsewhere. Let $\tau$ denote the running time of one ANN query, including the $O(d)$ distance evaluations that it performs. Typically, one has a constant neighborhood size $|N(A)|$ and a constant number of refreshed blocks per merge. Then, the algorithm issues $O(m)$ ANN queries, the exact scoring of the returned neighbors costs $O(|N(A)|d) = O(d)$ time per query, and heap maintenance costs $O(\log m)$ time per merge. Hence, the total time complexity is reduced to $O\big(m(\tau + d + \log m)\big)$.

## C.1 MTR-R PLANNER COMPLEXITY

Let $G$ be the set of candidate values in the diagonal grid, $m$ the number of target mixtures, and $d$ the number of source domains. A Forward-TV test for a pair $(A, B)$ and a grid value $\alpha$ must verify the cap for every descendant $j \in A \cup B$, so it costs $O\big((|A| + |B|)d\big)$ time. The direct diagonal implementation performs at most $m - 1$ agglomeration rounds; when $k$ blocks remain, the $\binom{k}{2}$ pairs together scan $(k-1)m$ descendant targets per grid value, because each block is scanned by the $k-1$ pairs that contain it. Hence one round costs $O\big(|G|(k-1)md\big)$ and the total worst-case running time is $O\big(|G|m^3 d\big)$. The active block state requires $O(md)$ storage, including the serialized merge history, which contains at most $m - 1$ checkpoints.

# D BEST FEASIBLE CHECKPOINT FOR A FIXED PAIR AS A LINEAR PROGRAM

Constrained by the above feasibility test, finding the best feasible checkpoint for a fixed pair of target blocks becomes harder than Lemma 1. We first expose the linear structure in the nonsmooth TV that defines $L_{\mathrm{fwd}}$.

**Lemma 7** (Linear representation of the Forward-TV cost)**.** *For a target $p \in \Delta^{d-1}$ and an edge increment $u \in \mathbb{R}^d_{\geq 0}$ with budget $s = \|u\|_1 = \sum_i u_i$, define the polyhedral epigraph*

$$\mathcal{H}(u,p) = \left\{ h \in \mathbb{R}^d_{\geq 0} : h_i \geq u_i - sp_i, \quad h_i \geq -u_i + sp_i \quad \textit{for all } i \right\},$$

*whose defining inequalities are affine in $(u, p, s, h)$. Then*

$$L_{\mathrm{fwd}}(u,p) = \frac{1}{2} \min_{h \in \mathcal{H}(u,p)} \sum_i h_i,$$

*so the TV cost of an edge is the optimal value of a linear program in the auxiliary variables $h$.*

Let blocks $A$ and $B$ have current root checkpoints $x = q_A$ and $y = q_B$. For every descendant $j \in A$, let $b_j^A$ denote the Forward-TV contribution of the path strictly below $x$, excluding the edge from the zero checkpoint to $x$. Initially $x = p^{(j)}$ and $b_j^{\{j\}} = 0$. If a candidate parent checkpoint is $q$, then the full score of a target $j \in A$ after the merge is

$$b_j^A + L_{\mathrm{fwd}}(x - q, p^{(j)}) + L_{\mathrm{fwd}}(q, p^{(j)}), \tag{2}$$

and the analogous expression with $B$ and $y$ holds for $j \in B$. By applying Lemma 7 to turn Equation (2) into a linear form, a maximum-mass feasible parent checkpoint for $A$ and $B$ is obtained from the following linear program.

**Theorem 5** (Maximum-mass feasible parent checkpoint for a fixed pair)**.** *Let blocks $A$ and $B$ have current root checkpoints $x$ and $y$. The best feasible parent checkpoint for $A$ and $B$ is the optimal solution $q^\star$ of the following linear program in $q$ and the auxiliary variables $h$:*

$$\begin{aligned}
\max_{q,h} \quad & \sum_i q_i \\
\textit{subject to} \quad & 0 \leq q_i \leq \min(x_i, y_i) \quad \textit{for all } i, \\
& h_j^0 \in \mathcal{H}(q, p^{(j)}), \quad h_j^A \in \mathcal{H}(x - q, p^{(j)}) \quad (j \in A), \\
& b_j^A + \frac{1}{2} \sum_i \left( h_{j,i}^0 + h_{j,i}^A \right) \leq \tau_f \quad (j \in A), \\
& h_j^0 \in \mathcal{H}(q, p^{(j)}), \quad h_j^B \in \mathcal{H}(y - q, p^{(j)}) \quad (j \in B), \\
& b_j^B + \frac{1}{2} \sum_i \left( h_{j,i}^0 + h_{j,i}^B \right) \leq \tau_f \quad (j \in B).
\end{aligned}$$

# E ADDITIONAL EXPERIMENTAL RESULTS

## E.1 PAIRED-SEED PROTOCOL

We fix the data-sampling seed across all 128 target mixtures to keep the paired comparisons aligned. Within each paired run, the protocol aligns model initialization, data-sampling randomness, and the validation stream between MTR and `Strict`. The multiset-preserving executor then guarantees that each MTR terminal consumes exactly the same child-domain sample multiset as its paired `Strict` proxy across its shared path and private tail. Thus, the comparison holds the underlying token instances fixed and isolates checkpoint reuse and ingestion order; the integer seeds are 41, 42, 43, 922, 2026, and 4243.

## E.2 RUN-LEVEL PROXY AND REGMIX-FIT RESULTS

Table 4 reports the six-seed values underlying Table 3. `MTR-r` denotes the Forward-TV planner at $\tau_f = 0.20$. $L_1$ and maximum domain deviation compare each method's mean top-128 composition with its paired `Strict` composition, and Proxy Spearman compares the corresponding 128 observed proxy losses.

Table 4: Run-level execution and top-128 composition results.

| Seed | Method | Saving ↑ | GPU-h ↓ | $L_1$ ↓ | Max deviation ↓ | Proxy Spearman ↑ |
|---|---|---|---|---|---|---|
| 41 | `Strict` | 0.0% | 26.76 | **0** | **0** | **1** |
| 41 | `Star` | 0.0% | 26.76 | **0** | **0** | **1** |
| 41 | `MTR-a` | **70.0%** | **7.96** | 0.203334 | 8.68 | 0.831246 |
| 41 | `MTR-r` | 49.0% | 14.14 | **0.149259** | **7.01** | **0.958131** |
| 42 | `Strict` | 0.0% | 26.97 | **0** | **0** | **1** |
| 42 | `Star` | 0.0% | 26.97 | **0** | **0** | **1** |
| 42 | `MTR-a` | **70.0%** | **7.69** | 0.244940 | 9.43 | 0.863291 |
| 42 | `MTR-r` | 49.0% | 13.93 | **0.128436** | **4.09** | **0.960391** |
| 43 | `Strict` | 0.0% | 26.88 | **0** | **0** | **1** |
| 43 | `Star` | 0.0% | 26.88 | **0** | **0** | **1** |
| 43 | `MTR-a` | **70.0%** | **7.66** | 0.329306 | 14.39 | 0.824292 |
| 43 | `MTR-r` | 49.0% | 13.98 | **0.098949** | **3.40** | **0.960388** |
| 922 | `Strict` | 0.0% | 26.91 | **0** | **0** | **1** |
| 922 | `Star` | 0.0% | 26.91 | **0** | **0** | **1** |
| 922 | `MTR-a` | **70.0%** | **8.15** | **0.196992** | **8.25** | 0.867325 |
| 922 | `MTR-r` | 49.0% | 13.81 | 0.237523 | 11.58 | **0.924720** |
| 2026 | `Strict` | 0.0% | 27.04 | **0** | **0** | **1** |
| 2026 | `Star` | 0.0% | 27.04 | **0** | **0** | **1** |
| 2026 | `MTR-a` | **70.0%** | **8.11** | 0.275490 | 10.09 | 0.849402 |
| 2026 | `MTR-r` | 49.0% | 13.78 | **0.187667** | **6.22** | **0.947518** |
| 4243 | `Strict` | 0.0% | 27.02 | **0** | **0** | **1** |
| 4243 | `Star` | 0.0% | 27.02 | **0** | **0** | **1** |
| 4243 | `MTR-a` | **70.0%** | **7.97** | 0.275575 | 10.92 | 0.852973 |
| 4243 | `MTR-r` | 49.0% | 13.81 | **0.166940** | **6.81** | **0.958265** |

The run-level results reproduce the aggregate pattern: `MTR-r` improves top-128 composition fidelity and proxy-loss alignment over `MTR-a` in the paired evaluation. Because the coordinatewise common prefix is zero in this experiment, `Star` is analytically identical to `Strict` and is reported with the corresponding `Strict` values.

### E.3 Forward-TV-constrained DP reference

The Forward-TV-constrained DP provides a like-for-like optimum for `MTR-r` at $\tau_f = 0.20$. Table 5 pools $m \in \{4, 5\}$, $d \in \{4, 8\}$, and 20 random seeds per configuration, for a total of 240 instances. `MTR-r` achieves $27.57\%$ mean token savings compared with $28.80\%$ under the Forward-TV-constrained DP; the corresponding pooled relative cost gap is $1.72\%$. The benchmark remains small because the DP retains path-dependent checkpoint states for every feasible binary subtree and diagonal candidate $\alpha \in \{1, 0.95, \ldots, 0.10\}$. Larger instances require multiple hours of computation and hundreds of GiB of memory.

Table 5: Forward-TV-constrained token savings at $\tau_f = 0.20$. Entries pool $m \in \{4, 5\}$, $d \in \{4, 8\}$, and 20 random seeds per configuration.

| Mixture family | Forward-TV-constrained DP ↑ | `MTR-r` ↑ |
|---|---|---|
| Sparse Dirichlet | **21.20%** | **21.20%** |
| Uniform Dirichlet | **48.02%** | 44.65% |
| RegMix-like | **17.18%** | 16.87% |
| Overall mean | **28.80%** | 27.57% |

### E.4 COMPLETE PLANNER CAP SWEEP

Table 6 reports a planner-only cap sweep across Forward-TV thresholds $\tau_f$ on the 128 target mixtures, produced by the executable child-domain multiset planner. These results measure structural token savings and tree topology.

Table 6: Forward-TV threshold sweep on the executable Essential-Web planner.

| $\tau_f$ | Saving ↑ | Merges | Roots | Time (s) |
|---|---|---|---|---|
| 0.10 | 32.3% | 121 | 7 | 321.54 |
| 0.20 | 49.0% | 124 | 4 | 324.63 |
| 0.30 | **59.5**% | 123 | 5 | 123.65 |

Relaxing $\tau_f$ lowers the executable token cost. The default $\tau_f = 0.20$ yields $49.0\%$ token savings.

### E.5 SYNTHETIC FORWARD-TV SENSITIVITY

Forward-TV provides a direct operating point between shared-path composition control and reuse efficiency. Table 7 shows that every evaluated `MTR-r` plan satisfies its prescribed threshold. With $\tau_f = 0.10$, token savings range from $24.11\%$ to $63.01\%$ across the three representative families. Increasing the threshold to $0.20$ raises this range to $41.33\%$–$75.17\%$, while $\tau_f = 0.30$ yields $53.43\%$–$79.84\%$. A larger threshold enlarges the feasible reuse set and yields greater token savings on all evaluated families.

Table 7: Forward-TV threshold sensitivity on the three representative families defined in Table 1 ($m = 300$, $d = 8$). Entries are mean token savings over six random seeds. The Overall mean is computed before rounding, and every plan satisfies $\max_j F_j \le \tau_f$.

| Synthetic family | $\tau_f = 0.10$ ↑ | $\tau_f = 0.20$ ↑ | $\tau_f = 0.30$ ↑ |
|---|---|---|---|
| Sparse Dirichlet | **63.01**% | **75.17**% | **79.84**% |
| Uniform Dirichlet | 43.97% | 66.33% | 75.10% |
| RegMix-like | 24.11% | 41.33% | 53.43% |
| Overall mean | 43.69% | 60.94% | 69.46% |

### E.6 ADDITIONAL SYNTHETIC RESULTS

Unless stated otherwise, family-level results are averaged over random seeds. Each reported Overall mean assigns equal weight to the listed families and is computed from their unrounded family-level values. Table 8 lists the eleven synthetic mixture families. Tables 9 to 11 report per-family token savings versus $m$; Tables 12 to 14 report the corresponding $d$-scans.

For a sparse family with sparsity $s$, each target retains a uniformly selected support of size $\max(1, \mathrm{round}((1-s)d))$ and draws symmetric $\mathrm{Dirichlet}(\mathbf{1})$ weights on the active coordinates. For a clustered family, each center is drawn from $\mathrm{Dirichlet}(\mathbf{1}_d)$, a center is chosen uniformly for every target, and the target is drawn from a Dirichlet distribution with parameter vector equal to the stated concentration times that center, plus $10^{-6}$ per coordinate for numerical positivity. The one-hot family selects its active coordinate uniformly. The logistic-normal family draws independent standard-normal logits and applies softmax. The multinomial family normalizes counts from $\mathrm{Multinomial}(100, \mathbf{1}_d/d)$. The stick-breaking family draws $d-1$ breaks from $\mathrm{Beta}(1,1)$ and randomly permutes the resulting coordinates. The RegMix-like family draws $c \sim \mathcal{U}(0.1, 5)$ and then $p \sim \mathrm{Dirichlet}(c\pi)$ for uniform $\pi$.

### E.7 ANN NEIGHBORHOOD SIZE FOR MTR-A

Table 15 varies the HNSW neighborhood size $k$ for greedy MTR-a. On one-hot targets at $m = 300$, $k = 3$ raises the cost ratio from $0.027$ to $0.743$, while $k \ge 5$ matches exact-pair search. Across

Table 8: Synthetic mixture families. Each family induces a distinct geometry of overlap among target mixtures, ranging from dense symmetric weights to sparse, clustered, and RegMix-like compositions.

| Code name | Display name | Construction | Parameters |
|---|---|---|---|
| `uniform_sym` | Symmetric Dirichlet | $\mathrm{Dirichlet}(\alpha \mathbf{1}_d)$ | $\alpha = 5.0$ |
| `uniform_peaked` | Peaked Dirichlet | $\mathrm{Dirichlet}(\alpha \mathbf{1}_d)$ | $\alpha = 1.0$ |
| `sparse_mild` | Sparse Dirichlet (mild) | sparse Dirichlet | sparsity $0.5$, $\alpha = 1.0$ |
| `sparse_heavy` | Sparse Dirichlet (heavy) | sparse Dirichlet | sparsity $0.8$, $\alpha = 1.0$ |
| `clustered_quad_mid` | Clustered-4 (medium) | clustered Dirichlet | 4 centers, concentration $50$, $\alpha = 1.0$ |
| `clustered_oct_loose` | Clustered-8 (loose) | clustered Dirichlet | 8 centers, concentration $20$, $\alpha = 1.0$ |
| `onehot` | One-hot | random one-hot | — |
| `logitnormal` | Logistic-normal | logistic-normal | logit scale $1.0$ |
| `multinomial` | Multinomial grid | multinomial grid | resolution $100$ |
| `stickbreaking` | Stick-breaking | stick-breaking | $\alpha = 1.0$ |
| `regmix` | RegMix-like | $c \sim \mathcal{U}(0.1, 5)$, $p \sim \mathrm{Dirichlet}(c\pi)$ | uniform prior $\pi$ |

Table 9: Per-family token savings (%) for `MTR-a` as $m$ increases at $d = 8$, averaged over three random seeds.

| Family | 50 | 100 | 200 | 300 | 500 | 1000 | 2000 |
|---|---|---|---|---|---|---|---|
| Clustered-8 (loose) | 61.6 | 67.5 | 71.6 | 74.2 | 76.4 | 79.2 | 81.5 |
| Clustered-4 (medium) | 75.0 | 78.9 | 81.7 | 82.9 | 84.6 | 86.3 | 87.7 |
| Logistic-normal | 59.0 | 62.8 | 67.0 | 69.1 | 71.8 | 74.6 | 77.1 |
| Multinomial grid | 82.8 | 84.3 | 85.6 | 86.6 | 87.5 | 88.7 | 89.9 |
| One-hot | 84.0 | 92.0 | 96.0 | 97.3 | 98.4 | 99.2 | 99.6 |
| Sparse Dirichlet (heavy) | 63.2 | 74.4 | 80.6 | 84.0 | 87.8 | 91.8 | 94.7 |
| Sparse Dirichlet (mild) | 53.5 | 60.2 | 65.5 | 68.8 | 72.4 | 76.5 | 80.1 |
| Stick-breaking | 55.4 | 63.7 | 69.8 | 72.4 | 75.2 | 78.8 | 81.6 |
| Peaked Dirichlet | 57.2 | 61.2 | 64.9 | 67.1 | 69.5 | 72.5 | 75.2 |
| Symmetric Dirichlet | 73.5 | 76.3 | 78.6 | 79.9 | 81.4 | 83.1 | 84.7 |
| RegMix-like | 52.5 | 59.4 | 65.5 | 67.9 | 71.4 | 75.1 | 78.3 |
| Overall mean | 65.2 | 71.0 | 75.2 | 77.3 | 79.7 | 82.4 | 84.6 |

the sparse, uniform, and clustered families at $m = 300$, every reported HNSW result differs from exact-pair search by at most $0.001$. At $m = 100$, all reported one-hot settings attain the same $0.080$ cost ratio.

Table 10: Per-family token savings (%) for `MTR-r` as $m$ increases at $d = 8$, averaged over three random seeds at $\tau_f = 0.20$.

| Family | 50 | 100 | 200 | 300 | 500 | 1000 | 2000 |
|---|---|---|---|---|---|---|---|
| Clustered-8 (loose) | 37.5 | 44.8 | 49.8 | 53.5 | 57.2 | 61.6 | 66.1 |
| Clustered-4 (medium) | 57.8 | 62.7 | 67.5 | 69.4 | 72.6 | 76.1 | 79.1 |
| Logistic-normal | 33.7 | 37.5 | 42.7 | 45.7 | 49.5 | 53.2 | 57.6 |
| Multinomial grid | 72.6 | 75.2 | 76.8 | 78.7 | 80.6 | 82.8 | 84.9 |
| One-hot | 84.0 | 92.0 | 96.0 | 97.3 | 98.4 | 99.2 | 99.6 |
| Sparse Dirichlet (heavy) | 46.0 | 60.2 | 70.0 | 75.4 | 81.5 | 88.0 | 92.5 |
| Sparse Dirichlet (mild) | 27.8 | 34.4 | 38.8 | 42.9 | 48.5 | 55.0 | 61.9 |
| Stick-breaking | 29.1 | 37.7 | 44.4 | 48.1 | 52.5 | 58.4 | 64.0 |
| Peaked Dirichlet | 33.6 | 37.1 | 40.7 | 43.2 | 46.1 | 50.7 | 54.7 |
| Symmetric Dirichlet | 55.5 | 60.2 | 63.7 | 66.4 | 68.9 | 72.1 | 75.2 |
| RegMix-like | 26.7 | 32.2 | 37.9 | 41.2 | 45.9 | 51.9 | 57.3 |
| Overall mean | 45.8 | 52.2 | 57.1 | 60.2 | 63.8 | 68.1 | 72.1 |

Table 11: Per-family token savings (%) for `Star` as $m$ increases at $d = 8$, averaged over three random seeds.

| Family | 50 | 100 | 200 | 300 | 500 | 1000 | 2000 |
|---|---|---|---|---|---|---|---|
| Clustered-8 (loose) | 1.1 | 0.6 | 0.4 | 0.4 | 0.2 | 0.2 | 0.0 |
| Clustered-4 (medium) | 8.3 | 5.0 | 4.6 | 4.5 | 3.4 | 3.0 | 2.4 |
| Logistic-normal | 8.2 | 5.2 | 4.2 | 3.5 | 3.1 | 2.5 | 2.0 |
| Multinomial grid | 47.7 | 43.6 | 35.8 | 34.6 | 30.9 | 26.6 | 22.3 |
| One-hot | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 |
| Sparse Dirichlet (heavy) | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 |
| Sparse Dirichlet (mild) | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 |
| Stick-breaking | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 |
| Peaked Dirichlet | 3.4 | 1.1 | 0.5 | 0.4 | 0.2 | 0.1 | 0.1 |
| Symmetric Dirichlet | 27.4 | 22.7 | 19.2 | 17.7 | 15.8 | 13.8 | 12.8 |
| RegMix-like | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 |
| Overall mean | 8.7 | 7.1 | 5.9 | 5.6 | 4.9 | 4.2 | 3.6 |

Table 12: Per-family token savings (%) for `MTR-a` as domain dimensionality increases at $m = 300$, averaged over three random seeds.

| Family | 4 | 8 | 10 | 16 | 32 | 64 | 128 |
|---|---|---|---|---|---|---|---|
| Clustered-8 (loose) | 88.1 | 74.2 | 69.5 | 58.9 | 43.5 | 31.9 | 21.9 |
| Clustered-4 (medium) | 92.5 | 82.9 | 79.7 | 72.1 | 58.2 | 44.1 | 32.5 |
| Logistic-normal | 85.1 | 69.1 | 65.1 | 56.8 | 48.2 | 42.6 | 39.5 |
| Multinomial grid | 95.0 | 86.6 | 83.2 | 75.0 | 61.0 | 46.3 | 32.5 |
| One-hot | 98.7 | 97.3 | 96.7 | 94.7 | 89.3 | 78.9 | 62.3 |
| Sparse Dirichlet (heavy) | 98.7 | 84.0 | 80.9 | 64.1 | 40.9 | 25.5 | 17.0 |
| Sparse Dirichlet (mild) | 92.6 | 68.8 | 61.8 | 47.8 | 34.2 | 26.1 | 21.6 |
| Stick-breaking | 85.2 | 72.4 | 68.6 | 62.8 | 55.2 | 46.0 | 37.0 |
| Peaked Dirichlet | 84.1 | 67.1 | 62.1 | 54.4 | 46.4 | 41.7 | 38.8 |
| Symmetric Dirichlet | 89.5 | 79.9 | 77.5 | 73.5 | 69.3 | 67.0 | 65.4 |
| RegMix-like | 84.7 | 67.9 | 63.5 | 54.7 | 44.7 | 35.3 | 27.0 |
| Overall mean | 90.4 | 77.3 | 73.5 | 65.0 | 53.7 | 44.1 | 36.0 |

Table 13: Per-family token savings (%) for `MTR-r` as domain dimensionality increases at $m = 300$, averaged over three random seeds at $\tau_f = 0.20$.

| Family | 4 | 8 | 10 | 16 | 32 | 64 | 128 |
|---|---|---|---|---|---|---|---|
| Clustered-8 (loose) | 81.0 | 53.5 | 45.7 | 31.2 | 18.9 | 13.7 | 10.3 |
| Clustered-4 (medium) | 88.8 | 69.4 | 62.7 | 48.0 | 29.2 | 19.2 | 14.0 |
| Logistic-normal | 75.2 | 45.7 | 39.2 | 28.9 | 22.2 | 18.9 | 17.2 |
| Multinomial grid | 93.9 | 78.7 | 71.6 | 55.2 | 33.7 | 20.1 | 12.8 |
| One-hot | 98.7 | 97.3 | 96.7 | 94.7 | 89.3 | 78.9 | 62.3 |
| Sparse Dirichlet (heavy) | 98.7 | 75.4 | 69.8 | 35.7 | 14.7 | 10.0 | 8.3 |
| Sparse Dirichlet (mild) | 89.5 | 42.9 | 32.3 | 19.9 | 13.9 | 11.4 | 10.2 |
| Stick-breaking | 74.3 | 48.1 | 41.4 | 34.2 | 27.7 | 22.2 | 17.9 |
| Peaked Dirichlet | 73.2 | 43.2 | 36.4 | 27.7 | 21.7 | 19.0 | 17.5 |
| Symmetric Dirichlet | 84.4 | 66.4 | 61.2 | 53.0 | 45.7 | 41.1 | 38.2 |
| RegMix-like | 73.5 | 41.2 | 35.5 | 25.5 | 19.6 | 14.8 | 11.9 |
| Overall mean | 84.7 | 60.2 | 53.9 | 41.3 | 30.6 | 24.5 | 20.1 |

Table 14: Per-family token savings (%) for `Star` as domain dimensionality increases at $m = 300$, averaged over three random seeds.

| Family | 4 | 8 | 10 | 16 | 32 | 64 | 128 |
|---|---|---|---|---|---|---|---|
| Clustered-8 (loose) | 0.1 | 0.4 | 0.1 | 0.0 | 0.0 | 0.0 | 0.0 |
| Clustered-4 (medium) | 2.7 | 4.5 | 3.1 | 1.2 | 0.2 | 0.1 | 0.0 |
| Logistic-normal | 3.1 | 3.5 | 3.5 | 3.5 | 3.6 | 3.7 | 3.7 |
| Multinomial grid | 53.2 | 34.6 | 27.9 | 10.3 | 0.0 | 0.0 | 0.0 |
| One-hot | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 |
| Sparse Dirichlet (heavy) | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 |
| Sparse Dirichlet (mild) | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 |
| Stick-breaking | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 |
| Peaked Dirichlet | 0.4 | 0.4 | 0.4 | 0.4 | 0.3 | 0.3 | 0.3 |
| Symmetric Dirichlet | 18.3 | 17.7 | 17.6 | 17.9 | 17.9 | 18.2 | 18.2 |
| RegMix-like | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 |
| Overall mean | 7.1 | 5.6 | 4.8 | 3.0 | 2.0 | 2.0 | 2.0 |

Table 15: ANN versus exact-pair `MTR-a` at $d = 8$. Entries are mean cost ratios over six random seeds.

| Family | $m$ | Exact-pair `MTR-a` ↓ | HNSW, $k = 3$ ↓ | HNSW, $k = 5$ ↓ | HNSW, $k = 10$ ↓ |
|---|---|---|---|---|---|
| One-hot | 100 | **0.080** | **0.080** | **0.080** | **0.080** |
| One-hot | 300 | **0.027** | 0.743 | **0.027** | **0.027** |
| Sparse Dirichlet | 300 | **0.161** | **0.161** | **0.161** | **0.161** |
| Uniform Dirichlet | 300 | 0.200 | **0.199** | 0.200 | 0.200 |
| Clustered Dirichlet | 300 | **0.174** | **0.174** | **0.174** | **0.174** |

### E.8 Data and execution details

The 10 parent domains are Arts; General works, books and libraries, and information sciences; History and Geography; Industrial arts, Technology, and Engineering; Literature; Philology, or Language and languages; Philosophy and psychology; Religion; Science and Natural history; and Social sciences.

We pin Essential-Web to Hugging Face revision `ce4eccc7e960` and construct the sampling frame from 900 unique Parquet shards spanning 40 crawl partitions. The scan contains 74,479,497 source documents. We retain documents with a nonempty FDC label, English confidence of at least $0.70$, at least 128 tokenizer tokens, no detected extraction artifact or missing content, and zero Lorem Ipsum and UT1-blacklist indicators. We exclude documents labeled as spam or advertisements, truncated, or other/unclassified. These filters retain 17,946,960 documents and 21.564B tokens before constructing the proxy-training pool and evaluation stream.

We assign each document deterministically to train, validation, or test using a hash of its document identifier with a $90/5/5$ split. The proxy-training pool contains 812,422 unique document identifiers, whereas the fixed validation selection contains 3,587 unique document identifiers and 4,194,304 packed tokens. Neither selection contains duplicate identifiers, and their identifier intersection is empty. Each selection manifest retains the source crawl, shard, row group, and document identifier, permitting an audit without redistribution of webpage text.

Essential-Web is derived from Common Crawl data. Its dataset card makes the database contributions available under the ODC Attribution License 1.0 and directs users to comply with the Common Crawl Terms of Use. The database license does not replace copyright, privacy, or other rights that may apply to individual pages. We report only aggregate results and do not redistribute raw webpage text in the submission artifacts.

Tree planning, token accounting, and data materialization use the same 128 target mixtures in the 81-child-domain representation, and the resulting counts are aggregated deterministically into the 10 parent domains for RegMix surrogate fitting and evaluation. The planner evaluates $\alpha \in \{1.00, 0.95, \ldots, 0.10\}$ and stops when no positive-gain feasible merge remains. The archived configurations specify a $0.98$M-parameter decoder-only proxy, 7,629 optimization steps, approximately 1B tokens per proxy, AdamW, a cosine learning-rate schedule, BF16 arithmetic, and compiled execution. Within each paired run, all methods use the same model initialization, sampling randomness, and validation stream. The multiset-preserving executor guarantees that each MTR terminal's shared path and private tail jointly consume exactly the same child-domain sample multiset as its paired `Strict` proxy.

The RegMix analysis uses an `LGBMRegressor` with 1,000 estimators, learning rate $0.01$, 31 leaves, and a fixed surrogate seed. The 100,000-mixture candidate procedure samples $c \sim \mathrm{Uniform}[0.1, 5.0]$ and then $p \sim \mathrm{Dirichlet}(c\pi)$. The same stored candidate pool is used for every paired surrogate comparison.

## F Additional figures

Figures 2 to 4 collect the illustrative constructions that the main text references but do not display.

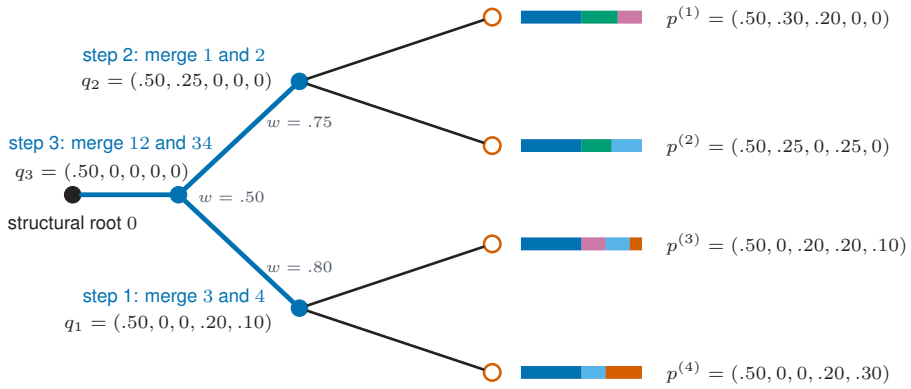



Figure 2: A concrete run of greedy agglomeration. The leaves are four target mixtures, and each internal checkpoint (in blue) is their coordinatewise minimum.

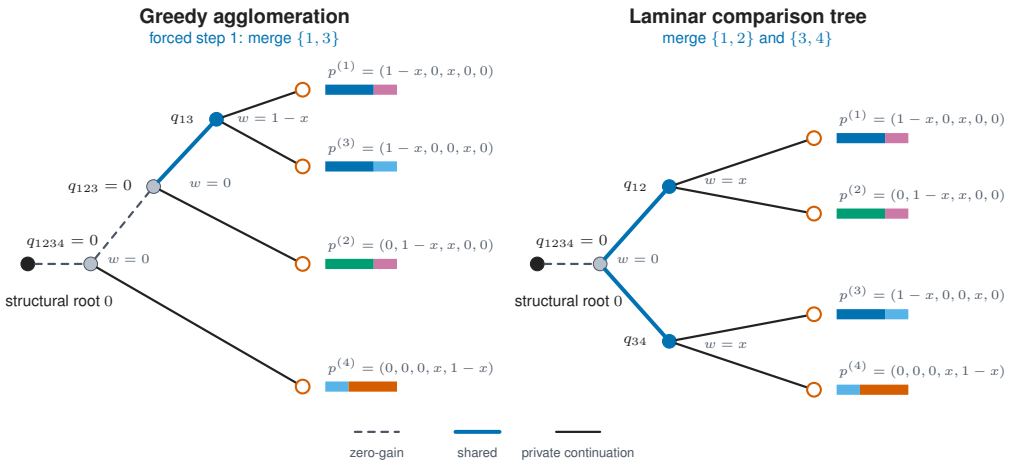



Figure 3: The four-target construction behind the tight $1/2$-approximation guarantee. Greedy agglomeration is forced to merge targets 1 and 3, whose overlap is $1 - x$; every later merge then has weight zero. The laminar comparison tree instead merges 1 with 2 and 3 with 4, obtaining total reuse gain $2x$. As $x \to 1/2$, the ratio $(1 - x)/(2x)$ approaches $1/2$.

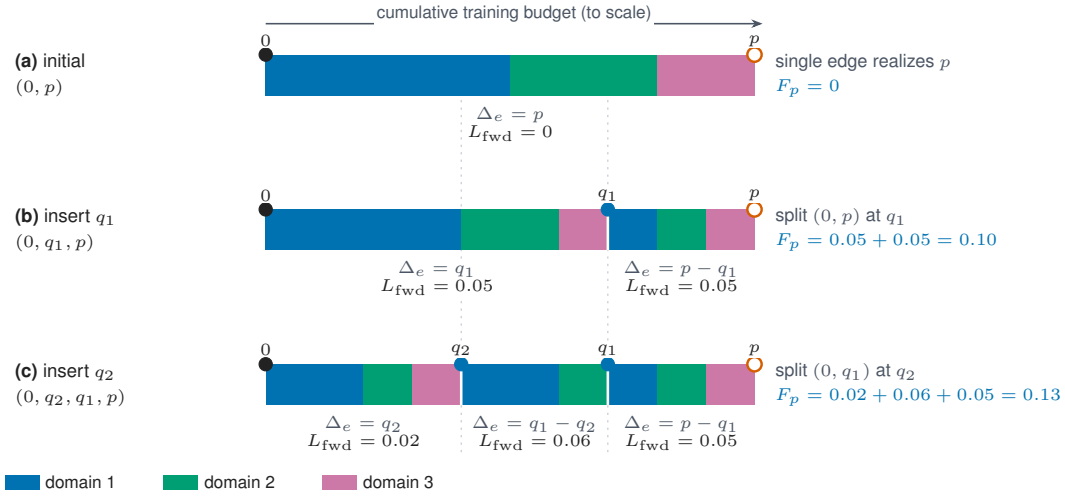



Figure 4: Forward-TV accounting for a single target $p$ as checkpoints are inserted above it. The path is refined from $(0, p)$ to $(0, q_1, p)$ and then to $(0, q_2, q_1, p)$, where each insertion replaces an edge by two edges. The score $F_p$ accumulates these contributions along the path. The displayed values use $p = (0.5, 0.3, 0.2)$, $q_1 = (0.4, 0.2, 0.1)$, and $q_2 = (0.2, 0.1, 0.1)$.