TRM-Planner: Offline Target Planning and Distillation for Tiny Recursive Models

Anonymous ACL submission

Abstract

Tiny Recursive Models (TRMs) perform iterative reasoning with an Adaptive Computation Time (ACT)-style loop, but their supervised training targets can be brittle and their halting behavior can be difficult to tune. We introduce **TRM-Planner**, a two-stage teacher-cache distillation recipe that shifts compute to an *offline* teacher-cache stage. A frozen TRM checkpoint is unrolled for multiple refinement steps and stochastic rollouts; for each instance we cache a small set of *teacher entries* (tokens, logits, step index, and quality metadata). A student TRM is then trained with the standard TRM objective plus a distillation loss computed from cached entries. Across Sudoku-Extreme and ARC-AGI-1/2, TRM-Planner shows an improvement over our reproduced TRM baseline while leaving student-time inference unchanged. On ARC1/ARC2 with 7M parameters, the two-attempt accuracy (`pass@2`) increases from 43.1%→**48.1%** and 6.7%→**9.2%**, respectively.

1 Introduction

Many grid-based reasoning tasks can be cast as fixed-layout sequence prediction, enabling non-autoregressive parallel decoding (Gu et al., 2018; Lee et al., 2018; Ghazvininejad et al., 2019). Tiny Recursive Models (TRMs) iteratively refine a full output sequence with an ACT-style loop (Jolicoeur-Martineau, 2025; Graves, 2016; Dehghani et al., 2019). In standard TRM training, supervision is effectively tied to a single refinement depth chosen by the learned halting policy; if halting is miscalibrated, the resulting targets can be brittle and require tuning. We propose **TRM-Planner**, a two-stage teacher-cache distillation recipe that improves supervision by caching higher-quality teacher targets offline and distilling from them.

Instead of changing the TRM architecture, we run a frozen TRM checkpoint (teacher) multiple times for up to T_{max} refinement steps while *ignoring* its halting decision; we then select high-quality outputs and, when available, a small set of non-redundant teacher logits to store as teacher entries. The student then trains on ground-truth labels plus a distillation loss from the cached entries. This yields a practical "pay once, reuse many" approach: expensive compute is spent offline once per dataset recipe, and then amortized over multiple training runs (Figure 1).

Our contributions are summarized as follows:

a) **TRM-Planner (teacher-cache distillation).** We propose a two-stage teacher-cache distillation recipe for TRMs: (i) we use an offline planner that unrolls a frozen TRM teacher across refinement steps/rollouts and caches high-quality intermediate targets; (ii) we use student training that augments the standard TRM objective with a weighted hard or soft distillation loss from the cache.

b) **Label-aware target planning and optional diversity.** We introduce a supervised planner score and a stop/continue beam search over refinement depth to select the teacher step. For multi-entry caches, we optionally apply STABLEJS-MMR, a StableMax-JS MMR selector on supervised tokens; empirically, diversity can collapse or be non-monotonic on near-unique-output tasks (e.g., Sudoku). We treat it as a tunable heuristic.

c) **StableMax-space distillation for TRMs.** We implement stable distillation by using supervised-token masking coupled with per-example normalization, which computes the KL loss in StableMax probability space to match TRM's StableMax likelihood.

d) **Evaluation and cost characterization.** On Sudoku-Extreme and ARC-AGI-1/2, we show consistent gains over reproduced TRM baselines at

fixed student-time inference; this includes cache-only and compute-matched controls (ARC1), and provides offline compute/storage costs.

2 Background and Preliminaries

Adaptive computation and iterative refinement. Adaptive Computation Time (ACT) provides a recurrent computation with a learned halting rule, thus allowing the number of refinement steps to vary across examples up to a fixed cap (Graves, 2016). Our base model is a Tiny Recursive Model (TRM), a lightweight recurrent refinement architecture trained for grid-based reasoning (Jolicoeur-Martineau, 2025). More broadly, TRM's non-autoregressive "refine-in-parallel" loop is related to iterative refinement methods in non-autoregressive sequence modeling (Lee et al., 2018; Ghazvininejad et al., 2019; jan).

Offline distillation. Knowledge distillation trains a student to match a teacher's outputs (Bucilua et al., 2006; Hinton et al., 2015; Ba and Caruana, 2014; Romero et al., 2015; Kim and Rush, 2016; Furlanello et al., 2018; Xie et al., 2020). In our setting the teacher supervision is generated *offline*: we cache teacher predictions once and reuse them throughout student training.

3 Tiny Recursive Model (TRM)

Problem setting. We represent grid-based reasoning as fixed-length sequence prediction. Each instance consists of an input token sequence $x \in \mathcal{V}^L$ and labels $y \in (\mathcal{V} \cup \{\texttt{IGNORE}\})^L$, where IGNORE denotes positions that are not supervised. Let the supervised index set be

$$M = \{i \in \{1, \dots, L\} : y_i \neq \texttt{IGNORE}\}. \quad (1)$$

TRM as *inner* refinement + *outer* ACT loop. TRM is a non-autoregressive iterative model. We conceptually distinguish two components: (i) a *TRM inner* refinement operator, and (ii) an *outer* loop that repeatedly applies this operator.

TRM inner. Let c_{t-1} denote the carry state at step $t-1$ (an internal representation maintained across steps). The TRM inner operator produces updated carry and predictions:

$$(c_t,\ z_t,\ q_t^{\text{halt}}) = \text{TRMInner}_\theta(c_{t-1}, x). \quad (2)$$

where $z_t \in \mathbb{R}^{L \times |\mathcal{V}|}$ are token logits and q_t^{halt} parameterizes a learned halting rule.

In our formulation, the carry is a pair of latent states $c_t = (z_t^H, z_t^L)$ that are carried across ACT steps. TRMInner corresponds to one call to the TRM's *deep recursion* operator: it performs H_{cycles} outer recursion cycles, and within each outer cycle it runs L_{cycles} *latent recursion* updates of z^L with input injection, followed by an update of z^H. In TRM, the first $H_{\text{cycles}}-1$ outer cycles are executed without gradient update, and we backpropagate only through the final outer cycle. The returned carry is detached before the next ACT step, while the heads read out token logits from z_t^H and halting logits from the first (puzzle-embedding) position.

Outer ACT loop. Starting from an initialization $c_0 = \text{init}(x)$, the outer loop applies (2) for up to $T_{\max}$ steps. We define the step-t prediction token-wise by $\hat{y}_i^{(t)} = \arg\max_{v \in \mathcal{V}} z_{t,i,v}$ for $i = 1, \dots, L$. During *training*, the model uses the halting head to decide whether an example should stop being refined and be replaced by a new example in the same batch slot (an ACT-style "streaming" schedule). During *evaluation*, we run a fixed number of steps $T_{\max}$ and report metrics from the final step; this avoids conflating accuracy with halting calibration.

StableMax token likelihood. The TRM maps logits to probabilities using StableMax (Prieto et al., 2025), an alternative normalization that improves numerical stability when logits become large in magnitude.

For $u \in \mathbb{R}^{|\mathcal{V}|}$ and a small constant $\varepsilon > 0$ (for numerical stability),

$$s(u_j) = \begin{cases} u_j + 1, & u_j \geq 0 \\ \frac{1}{1 - u_j + \varepsilon}, & u_j < 0 \end{cases} \qquad p_j = \frac{s(u_j)}{\sum_k s(u_k)}. \quad (3)$$

The StableMax cross-entropy at supervised position i is $\text{CE}_{\text{stable}}(z_{t,i}, y_i) = -\log p_{y_i}$.

Exact-match metric. For a chosen evaluation step t_{eval} (in our experiments, $t_{\text{eval}} = T_{\max}$), we report the exact match over supervised positions:

$$\text{ExactAcc} = \mathbb{E}\Big[\mathbb{I}\Big(\forall i \in M,\ \hat{y}_i^{(t_{\text{eval}})} = y_i\Big)\Big]. \quad (4)$$

Training objective. TRM optimizes a supervised token loss and a halting loss that teaches the halting head to predict whether the current prediction is

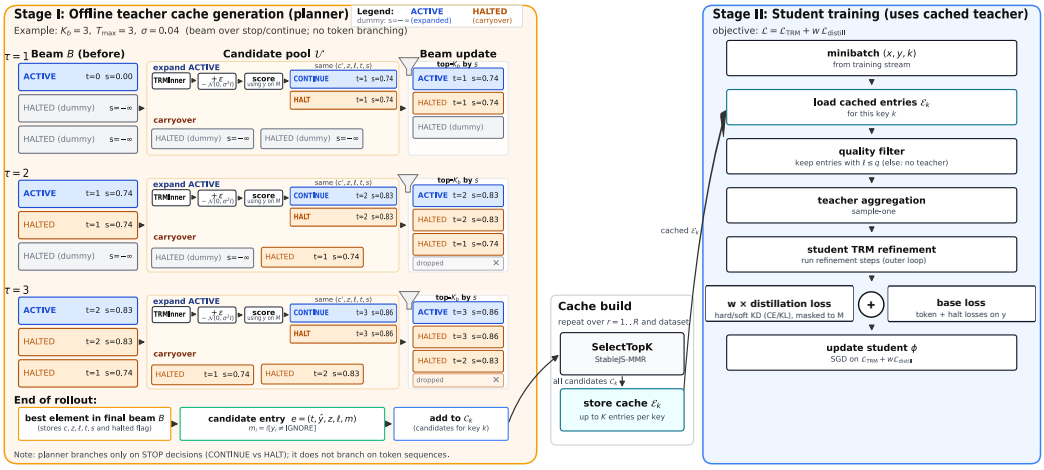

Figure 1: **TRM-Planner training framework overview. Stage I (offline teacher cache generation).** A frozen TRM teacher is unrolled up to $T_{\max}$ steps (optionally with R stochastic rollouts and a beam over stop/continue decisions) while ignoring the teacher's learned halting. Candidate steps are scored using supervised tokens and used to build a small cache of teacher entries per key with diversity-aware cache selection using StableJS-MMR. **Stage II (student training with cached teacher).** The student retrieves cached entries, optionally filters by teacher quality, aggregates teacher targets (sample-one), and optimizes the base TRM objective plus a weighted distillation loss (hard/soft KD) applied on supervised tokens.

exactly correct. We define the per-step token loss

$$\mathcal{L}_{\text{tok}}(t) = \frac{1}{|M|} \sum_{i \in M} \text{CE}_{\text{stable}}(z_{t,i}, y_i), \quad (5)$$

and the step-t correctness indicator $r_t = \mathbb{I}(\forall i \in M,\ \hat{y}_i^{(t)} = y_i)$. The halting head is trained with a binary cross-entropy (and optionally a continuation auxiliary loss):

$$\mathcal{L}_{\text{halt}}(t) = \text{BCEWithLogits}(q_t^{\text{halt}},\ r_t). \quad (6)$$

Overall, TRM training minimizes

$$\mathcal{L}_{\text{TRM}} = \sum_{t=1}^{T_{\max}} \Big(\mathcal{L}_{\text{tok}}(t) + \lambda_{\text{halt}} \mathcal{L}_{\text{halt}}(t) \Big). \quad (7)$$

where λ_{halt} is used to weight the halting loss.

4 TRM-Planner: Offline Search for Higher-Quality Supervision

TRM-Planner is a two-stage framework that improves training targets without modifying the TRM architecture. Given a pretrained TRM checkpoint (*teacher*) θ, TRM-Planner performs an *offline* search over TRMInner trajectories to cache high-quality teacher entries. A student TRM ϕ is then trained with the standard TRM loss, plus an additional distillation term computed from the cached entries.

4.1 Offline candidate generation via step/trajectory selection

For each training instance (x, y), TRM-Planner generates multiple candidate teacher outputs by repeatedly applying the teacher's TRM inner operator while *ignoring* the teacher's own halting decision. Concretely, we unroll (2) for up to $T_{\max}$ steps and score each step by supervised loss and (optionally) halting confidence.

Stochastic rollouts. To expand the candidate pool, we run R independent rollouts. Each rollout starts from the same initialization c_0 and applies TRMInner repeatedly. Optionally, we inject Gaussian noise into the carry after each refinement step:

$$(c_t,\ z_t,\ q_t^{\text{halt}}) = \text{TRMInner}_\theta(c_{t-1}, x), \qquad c_t \leftarrow c_t + \epsilon_t, \qquad \epsilon_t \sim \mathcal{N}(0, \sigma^2 I). \quad (8)$$

where σ controls exploration. Importantly, the prediction head remains non-autoregressive; the planner does not branch on token choices.

Beam-style step search and the role of K_b (beam size). Within each rollout, we maintain a small beam over *stop/continue decisions* (not token-level branching). Each beam element stores (c, z, t) and a scalar planner score. At depth τ, each *active* element is advanced by one TRMInner_θ call and produces two candidates: (i) a *continue* state (kept

active) and (ii) a *halt* state (frozen and carried forward unchanged). We then keep the top-K_b candidates using a score calculation (Algorithm 1). Larger K_b explores more stopping choices but still yields a single best entry per rollout; note that this increases offline compute, but *not* the number of cached entries.

4.2 Planner score

Given logits z_t at step t, we compute a label-aware loss on supervised tokens,

$$\ell_t = \frac{1}{|M|} \sum_{i \in M} \mathrm{CE}_{\mathrm{stable}}(z_{t,i}, y_i), \qquad \mathrm{quality}_t = \exp(-\ell_t). \tag{9}$$

and optionally include a halting-confidence term $\mathrm{halt}_t = \mathrm{sigmoid}(q_t^{\mathrm{halt}})$. TRM-Planner assigns a scalar score:

$$\mathrm{score}(t) = \alpha\, \mathrm{quality}_t + \beta\, \mathrm{halt}_t - \lambda\, t, \tag{10}$$

where α, β, λ trade off supervised quality, halting confidence, and step length. This score is used *only* for selecting teacher entries during offline cache generation.

4.3 Teacher entries and cache keys

The output of the offline stage is a small set of teacher entries per cache key k. A cache key is typically the puzzle/task identifier. For datasets where a single task contains multiple distinct input–output pairs (e.g., ARC), we use an *example-level* key that distinguishes pairs within a task, to ensure cached teacher logits correspond to the same input x.

For each key k, we store up to K entries:

$$\mathcal{E}_k = \{e^{(1)}, \ldots, e^{(K)}\}, \qquad e^{(j)} = \big(t^{(j)},\ \hat{y}^{(j)},\ z^{(j)},\ \ell^{(j)},\ m_{\mathrm{sup}}^{(j)}\big). \tag{11}$$

where $t^{(j)}$ is the selected step, $\hat{y}^{(j)} = \arg\max z^{(j)}$, and $\ell^{(j)}$ is the supervised teacher loss on $M = \{i : y_i \neq \texttt{IGNORE}\}$. The cached mask $m_{\mathrm{sup}}^{(j)} \in \{0,1\}^L$ ("sup" = *supervised*) records which positions are supervised for that training instance: $m_{\mathrm{sup},i}^{(j)} = \mathbb{I}[y_i \neq \texttt{IGNORE}]$ (so it matches M). This mask is used only during *offline cache selection*: for example, StableJS-MMR (introduced in Section 4.4) computes logit-diversity (JS divergence) restricted to supervised positions. During student training, KD masking is determined by the current labels (and teacher availability), *not* by the cached mask. Caching logits enables soft distillation; caching tokens enables hard distillation.

4.4 Selecting top-K cached entries with StableJS-MMR

Naively selecting the top-K candidates by using a planner score can produce near-duplicate entries, especially when the task has essentially a unique correct output (e.g., Sudoku). We therefore use a diversity-aware selector in *logit space*, inspired by subset selection and diverse decoding (Carbonell and Goldstein, 1998; Vijayakumar et al., 2016; Kulesza and Taskar, 2012). We refer to this selector as StableJS-MMR.

Offline caching costs. Cache generation requires $O(RK_bT_{\max})$ teacher forward calls per example and can be storage-heavy when caching logits; we report observed ARC compute and cache sizes in Table 13 in the Appendix.

StableMax Jensen–Shannon divergence (diversity metric). Given two candidate logits $z^a, z^b \in \mathbb{R}^{L\times|\mathcal{V}|}$, let $P_i = \mathrm{StableMax}(z_i^a)$, $Q_i = \mathrm{StableMax}(z_i^b)$, and $S_i = \frac{1}{2}(P_i + Q_i)$. $\mathrm{JS}(P_i, Q_i) = \frac{1}{2}\mathrm{KL}(P_i \| S_i) + \frac{1}{2}\mathrm{KL}(Q_i \| S_i)$.

Let $m^a, m^b \in \{0,1\}^L$ be supervision masks and $m^{ab} = m^a \wedge m^b$ (elementwise AND).

Define $M_{ab} = \{i \in \{1,\ldots,L\} : m_i^{ab} = 1\}$.

$$d_{\mathrm{JS}}(a,b) = \begin{cases} \frac{1}{|M_{ab}|} \sum_{i \in M_{ab}} \mathrm{JS}(P_i, Q_i), & |M_{ab}| > 0, \\ 0, & |M_{ab}| = 0, \end{cases} \tag{12}$$

JS is not a training loss; it is used only as a proxy for cache selection.

Greedy MMR in StableMax-JS space. Let $\mathcal{C}_k$ denote all candidates accumulated for key k (across rollouts and occurrences). We select cached entries greedily. First choose the highest-score candidate $e^\star$. Then, for each subsequent entry, choose the candidate that trades off normalized score and novelty to the already-selected set:

$$e \leftarrow \arg\max_{c \in \mathcal{C}_k \setminus S} \Big[\lambda_{\mathrm{sel}}\, \widetilde{\mathrm{score}}(c) + (1-\lambda_{\mathrm{sel}}) \min_{s \in S} d_{\mathrm{JS}}(s,c)\Big]. \tag{13}$$

where S is the set of selected entries and $\widetilde{\mathrm{score}}$ is min–max normalized within $\mathcal{C}_k$.

Label access and inference-time behavior. Offline cache generation uses training labels (the supervised region M) only to score and select teacher entries. At evaluation time, we run the *student only* for a fixed $T_{\max}$ refinement steps and do not consult the cache or planner.

5 Student Training with Teacher Cache

5.1 Quality filtering and teacher aggregation

At training time, each example is associated with a cache key k and uses that to retrieve its cached entry set $\mathcal{E}_k$. We optionally apply a teacher-quality filter

$$\mathcal{E}'_k = \{e \in \mathcal{E}_k : \ell(e) \le q\}, \tag{14}$$

where q trades off coverage (i.e., how often teacher supervision is available) vs purity (i.e., how reliable the teacher targets are). If $\mathcal{E}'_k = \emptyset$, the example is trained using only the standard TRM supervision.

Sample-one: when multiple entries remain, we sample $e \sim \mathrm{Uniform}(\mathcal{E}'_k)$ per occurrence. Over repeated occurrences, this exposes the student to multiple teacher variants.

5.2 Hard and soft distillation losses

Student training minimizes the base TRM objective plus a distillation term:

$$\mathcal{L} = \mathcal{L}_{\mathrm{TRM}} + w \cdot \mathcal{L}_{\mathrm{distill}}, \tag{15}$$

with distillation weight $w \ge 0$.

Distillation mask (hard and soft). Let $M = \{i : y_i \neq \texttt{IGNORE}\}$ denote supervised positions. Distillation is applied only when a teacher entry is available for the example after quality filtering (i.e., the filtered entry set $\mathcal{E}'_k$ is non-empty). In that case, cached teacher entries provide targets (tokens and logits) for the full sequence length, so the distillation region reduces to the supervised region:

$$M_{\mathrm{KD}} = \begin{cases} M, & \text{if } \mathcal{E}'_k \neq \emptyset, \\ \emptyset, & \text{otherwise.} \end{cases} \tag{16}$$

If $M_{\mathrm{KD}} = \emptyset$, we set the distillation loss to zero for that example.

Which student refinement step receives KD? In our implementation, once an example has an available cached teacher entry, we apply the KD loss at *every* student refinement step while the example is active in the ACT-style training schedule, using teacher tokens/logits cached from the teacher's selected step. We store the teacher step index as metadata (the refinement depth at which logits/tokens were produced), but the student does not condition on it. Unless otherwise noted, $\mathcal{L}_{\mathrm{hard}}$ and $\mathcal{L}_{\mathrm{soft}}$ are computed per student refinement step and summed across steps, in the same way as $\mathcal{L}_{\mathrm{TRM}}$.

Hard distillation (token imitation).

$$\mathcal{L}_{\mathrm{hard}} = \frac{1}{|M_{\mathrm{KD}}|} \sum_{i \in M_{\mathrm{KD}}} \mathrm{CE}_{\mathrm{stable}}\big(z_i^{\mathrm{stu}}, \hat{y}_i^{\mathrm{tea}}\big), \quad \mathcal{L}_{\mathrm{hard}} = 0 \quad \text{if } |M_{\mathrm{KD}}| = 0. \tag{17}$$

Soft distillation (distribution matching). For temperature $T > 0$, a cached teacher entry provides logits z^{tea}. During student training, distillation is applied to the student logits emitted at the current refinement step for that example. We form teacher and student distributions at position i using StableMax:

$$P_T(\cdot|i) = \mathrm{StableMax}\big(z_i^{\mathrm{tea}}/T\big), \quad Q_T(\cdot|i) = \mathrm{StableMax}\big(z_i^{\mathrm{stu}}/T\big).$$

For a single teacher entry, the soft KD loss is

$$\mathcal{L}_{\mathrm{soft}} = T^2 \cdot \frac{1}{|M_{\mathrm{KD}}|} \sum_{i \in M_{\mathrm{KD}}} \mathrm{KL}(P_T(\cdot|i) \,\|\, Q_T(\cdot|i)), \quad \mathcal{L}_{\mathrm{soft}} = 0 \quad \text{if } |M_{\mathrm{KD}}| = 0. \tag{18}$$

Following standard distillation practice, we multiply by T^2 to keep gradient magnitudes comparable across temperatures (Hinton et al., 2015).

6 Algorithm Summary

TRM-Planner consists of (i) an offline cache generation with a frozen teacher, and (ii) an online student training with cached supervision. We summarize both stages in Algorithm 1 and Algorithm 2 using the notation from Sections 3 to 5.

7 Experiments

7.1 Datasets

Sudoku-Extreme. Following Jolicoeur-Martineau (2025), Sudoku-Extreme consists of extremely difficult 9×9 Sudoku puzzles in a small-sample regime: we train on 1,000 samples and evaluate on 423,000 test samples. As in TRM, we apply heavy rule-preserving augmentation, using 1,000 shuffling augmentations per training example. We represent each instance as a fixed-length token sequence with near-full supervision.

Algorithm 1 Stage I: Offline teacher cache generation

Input: teacher parameters θ; training set of pairs (x, y) with cache key k; rollouts R; planning horizon $T_{\max}$; beam size K_b; cache size K; noise σ; score weights (α, β, λ); selection weight λ_{sel}.
Initialize an empty candidate multiset $\mathcal{C}_k \leftarrow \emptyset$ for each cache key k.
for each training instance (x, y, k) **do**
  for $r = 1$ to R **do**
    Initialize beam $B \leftarrow \{(c_0, z{=}\varnothing, \ell{=}+\infty, t{=}0, s{=}0, \textit{halted}{=}\text{False})\}$.
    Add $K_b{-}1$ dummy *halted* states $(c_0, z{=}\varnothing, \ell{=}+\infty, t{=}0, s{=}-\infty, \textit{halted}{=}\text{True})$ to B.
    for $\tau = 1$ to $T_{\max}$ **do**
      Initialize an empty list of candidates $\mathcal{U} \leftarrow \emptyset$.
      for each beam state $(c, z, \ell, t, s, \textit{halted})$ in B **do**
        if *halted* **then**
          Add *carryover* candidate $(c, z, \ell, t, s, \textit{halted}{=}\text{True})$ to $\mathcal{U}$.
        else
          $(c', z, q^{\text{halt}}) \leftarrow \text{TRMInner}_\theta(c, x)$.
          if $\sigma > 0$ **then**
            $c' \leftarrow c' + \epsilon,\ \epsilon \sim \mathcal{N}(0, \sigma^2 I)$.
          end if
          Compute $\ell \leftarrow \frac{1}{|M|} \sum_{i \in M} \text{CE}_{\text{stable}}(z_i, y_i)$ and $\text{quality} \leftarrow \exp(-\ell)$.
          Compute score $s' \leftarrow \alpha\, \text{quality} + \beta\, \text{sigmoid}(q^{\text{halt}}) - \lambda\, \tau$.
          Add *continue* candidate $(c', z, \ell, \tau, s', \textit{halted}{=}\text{False})$ to $\mathcal{U}$.
          Add *halt* candidate $(c', z, \ell, \tau, s', \textit{halted}{=}\text{True})$ to $\mathcal{U}$.
        end if
      end for
      Update beam $B \leftarrow$ top-K_b elements of $\mathcal{U}$ by score.
    end for
    Let $(\bar{c}, \bar{z}, \bar{\ell}, \bar{t}, \bar{s}, \cdot)$ be the best-scoring element in B.
    Let $m \in \{0, 1\}^L$ be the supervised-region mask for this instance, i.e., $m_i = \mathbb{I}[y_i \neq \texttt{IGNORE}]$.
    Compute $\hat{y} \in \mathcal{V}^L$ where $\hat{y}_i = \arg\max_{v \in \mathcal{V}} \bar{z}_{i,v}$ for all i
    Create candidate entry $e = (\bar{t},\ \hat{y},\ \bar{z},\ \bar{\ell},\ m)$ and add to $\mathcal{C}_k$.
  end for
end for
for each cache key k **do**
  Select $\mathcal{E}_k \leftarrow \text{SelectTopK}(\mathcal{C}_k, K)$ using (13) (with d_{JS} from (12)).
  Store $\mathcal{E}_k$ for use during student training.
end for

Algorithm 2 Stage II: Student training with cached teacher data

Input: student parameters ϕ; training stream (x, y, k); cached entry sets $\{\mathcal{E}_k\}$; quality threshold q; distillation weight w; temperature T
for each minibatch of training instances **do**
  for each instance (x, y, k) in the minibatch **do**
    Retrieve entries $\mathcal{E}_k$ and optionally filter: $\mathcal{E}'_k = \{e \in \mathcal{E}_k : \ell(e) \leq q\}$.
    If $\mathcal{E}'_k = \emptyset$, set "no teacher" for this instance.
    Otherwise select teacher targets by sample-one (Section 5.1).
  end for
  For each training iteration, run one student refinement step (one ACT update applying TRMInner_ϕ).
  Compute $\mathcal{L}_{\text{TRM}}$ from (7).
  If teacher targets are available, compute $\mathcal{L}_{\text{distill}}$ for this refinement step and add it to the step loss.
    If using hard KD, compute (17). If using soft KD, compute (18) (with mask (16)) and temperature T.
  Update ϕ by minimizing $\mathcal{L} = \mathcal{L}_{\text{TRM}} + w\, \mathcal{L}_{\text{distill}}$.
end for

Maze-30x30-Hard (diagnostic only). Following Jolicoeur-Martineau (2025), the Maze-Hard problem consists of 30×30 mazes whose shortest path length exceeds 110, with 1,000 mazes in both train and test. Training uses 8 dihedral transformations per example. In our environment, Maze results were unstable; we therefore treat Maze as a diagnostic dataset and do not include it in headline claims (Appendix A).

ARC-AGI-1/2 (ARC1/ARC2). We evaluate on ARC-AGI-1 and ARC-AGI-2 using the **public evaluation sets** following Jolicoeur-Martineau (2025). Each task provides 2–3 input–output demonstration pairs and 1–2 test inputs; the maximum grid size is 30×30. Following the standard ARC convention, we report `pass@K`, with `pass@2` as the headline "two attempts" setting. ARC-AGI-1 contains 800 tasks and ARC-AGI-2 contains 1120 tasks. We additionally augment training with 160 ConceptARC tasks, and apply heavy augmentation (1,000 transforms per example: color permutations, dihedral transforms, and translations), as in TRM.

7.2 Metrics and evaluation protocol

For sequence-style datasets (Sudoku/Maze), we report token accuracy on supervised tokens and ExactAcc (Eq. 4), by doing evaluation at a fixed refinement step $t_{\text{eval}} = T_{\max}$ (no early stopping).

For ARC, we report `pass@K` (Chen et al., 2021). We generate multiple augmented predictions per test input, inverse-transform/crop them to the canonical grid, and then rank unique candidate outputs by vote count (tie-break: mean halting probability). `pass@K` checks whether the ground-truth output appears in the top-K ranked candidates, averaged over tasks; `pass@2` is the headline two-attempt setting.

7.3 Baselines and proposed method

In our experimental analysis, we compare the performance of the following approaches:
(i) **TRM:** the baseline Tiny Recursive Model trained only with ground-truth labels.
(ii) **TRM-Planner (search teacher):** offline generation of a teacher cache via label-aware step selection (and optional stochastic rollouts), followed by student training with distillation from the cache.
(iii) **TRM-KD rollout baseline (no planning):** an offline teacher-cache baseline that removes label-aware step selection. For each training example, we run the frozen teacher with a *single* forward refinement trajectory (no planner score, no beam over stop/continue decisions), and cache the teacher's output at the fixed evaluation step $t_{\text{eval}} = T_{\max}$ (tokens and logits). The student is then trained with the *same* distillation objective and the same cache-loading/filtering procedure as TRM-Planner. This isolates the contribution of label-aware step

Table 1: ARC results on the public ARC-AGI evaluation sets. $p@K$ denotes `pass@K` computed with the repository evaluator.

Method	KD	p@1 ↑	p@2 ↑	p@5 ↑	p@10 ↑	p@100 ↑	p@1000 ↑
ARC1							
TRM (reproduced baseline)	–	0.3825	0.4313	0.4988	0.5250	0.6075	0.6425
TRM (compute-matched)	–	0.4013	0.4513	0.5138	0.5588	**0.6500**	0.6900
TRM-KD (rollout, w=0.25)	soft	0.4075	0.4713	0.5263	0.5575	**0.6500**	0.6938
TRM-Planner (w=0.25)	soft	0.4063	0.4688	**0.5375**	0.5588	0.6475	0.6900
TRM-Planner (w=1.0)	soft	**0.4113**	**0.4813**	0.5238	0.5550	0.6425	**0.6950**
TRM-Planner (w=0.25)	hard	0.3988	0.4638	0.5200	**0.5613**	**0.6500**	0.6850
ARC2							
TRM (reproduced baseline)	–	0.0417	0.0667	**0.1069**	0.1111	0.1653	0.1875
TRM-Planner (w=0.25)	soft	**0.0792**	**0.0917**	**0.1069**	0.1181	**0.2069**	0.2181
TRM-Planner (w=0.25)	hard	0.0667	**0.0917**	0.1042	**0.1278**	0.1944	**0.2292**

selection from "distillation alone."
(iv) **Compute-matched TRM:** a TRM baseline trained longer (or resumed) to match the overall optimization budget, controlling for "just more SGD."

7.4 Cache and training setup

We use the TRM architecture and training objective from prior work. Unless otherwise stated, Student training uses sample-one aggregation and StableMax KD (Section 5). Both Teacher and student share the same architecture and initialize from the same checkpoint (self-distillation / continued training). Offline cache generation uses $(T_{\max}, K_b, R, K) = (16, 4, 8, 2)$ with $(\alpha, \beta, \lambda) = (1.0, 0.0, 0.01)$ and $(\lambda_{\text{sel}}, \sigma) = (0.7, 0.02)$. Distillation weight w varies depending on each run, and quality threshold q=0.5 and temperature T=1 for soft KD. For student-model training hyperparameters (optimizer, learning rate, batch size, etc.), we adopt the same configuration as TRM (Jolicoeur-Martineau, 2025). Full hyperparameters (optimizer, LR schedule, dataset-specific overrides), hardware, and seeds are described in Appendix B. Unless otherwise noted, we report results from a single run (seed 0) due to compute constraints.

Reproducibility note (TRM). We use the public TRM codebase (Jolicoeur-Martineau, 2025); Maze-Hard results were unstable in our environment, so we treat Maze as diagnostic (Appendix A).

7.5 Sudoku evaluation protocol and diagnostics

Several Sudoku configurations exhibit overshooting: the training fit can continue improving while test exact match degrades late in training. To avoid test-set feedback, **all hyperparameters reported in Table 3 (including w, q, T, σ) were fixed before any test-set evaluation**. Accordingly, we report **Final** (test exact accuracy at the end of a fixed, prespecified training budget) as our **primary** metric. Any mention of intermediate-checkpoint maxima ("Peak") is restricted to **post-hoc diagnostics**: we do *not* use Peak for model selection, early stopping, or hyperparameter tuning. For strict comparability, readers should focus on **Final** numbers in Table 3; post-hoc diagnostic sweeps are deferred to Appendix C.

7.6 Parameter Scaling Ablation on Sudoku

To evaluate whether TRM-Planner helps with smaller models, we vary the model width (hidden size) and measure performance as a function of parameter count. For each width, we train (i) a baseline TRM and (ii) a TRM-Planner student using a teacher of the *same* width (no larger teacher is used).

8 Results

Across Sudoku-Extreme and ARC (ARC1/ARC2), TRM-Planner consistently improves over the TRM baseline while keeping student-only inference unchanged.

ARC. On ARC-AGI-1 (ARC1; Table 1), cache-based distillation is a strong baseline: both the rollout-KD variant and the compute-matched TRM control are competitive with TRM-Planner. TRM-Planner achieves the best `pass@2` under w=1.0 (0.4813) and the best `pass@1000` (0.6950), but the gains over rollout-KD are modest, suggesting that

Table 2: **Context only:** pass@2 on the public ARC-AGI evaluation sets. "Reported" rows are adapted from Jolicoeur-Martineau (2025, Table 5); TRM rows are from our reproduced runs.

Dataset	Method	# Params	pass@2 ↑
ARC1	TRM-Planner (this work; best)	7M	**0.4813**
ARC1	TRM (reproduced baseline)	7M	0.431
ARC1	HRM (reported)	27M	0.403
ARC1	o3-mini-high (reported)	?	0.345
ARC1	Gemini 2.5 Pro (reported)	?	0.370
ARC1	Grok-4-thinking (reported)	1.7T	0.667
ARC2	TRM-Planner (this work; best)	7M	**0.0917**
ARC2	TRM (reproduced baseline)	7M	0.066
ARC2	HRM (reported)	27M	0.050
ARC2	o3-mini-high (reported)	?	0.030
ARC2	Gemini 2.5 Pro (reported)	?	0.049
ARC2	Grok-4-thinking (reported)	1.7T	0.160

Table 3: Sudoku-Extreme results (test split). Exact match on supervised tokens at a fixed evaluation step. We report **Final** as the primary metric.

Method	KD	w	ExactAcc (Final)↑
TRM (reproduced baseline)	–	–	0.6869
TRM (compute-matched)	–	–	0.7772
TRM-KD (rollout)	hard	0.25	0.8041
TRM-Planner (search)	hard	0.25	0.8269
TRM-Planner (search)	soft (StableMax)	0.25	**0.8277**

most improvement comes from cached teacher supervision rather than step selection alone.

On ARC-AGI-2 (ARC2; Table 1), TRM-Planner improves over the TRM baseline at small K (pass@2: 0.0667 → 0.0917). We do not include ARC2 rollout-KD or compute-matched controls due to compute constraints, so ARC2 should be interpreted as evidence for the end-to-end pipeline rather than a clean ablation of step selection.

Context and comparability. Table 2 reports pass@2 on the *public* ARC-AGI evaluation sets for context only. For large pretrained systems, we copy a subset of the two-attempt results reported by Jolicoeur-Martineau (2025, Table 5) (labeled "Chain-of-thought, pretrained" in that work). These numbers are not directly comparable to our direct-prediction TRM runs due to differences in model scale, prompting/scaffolding, and evaluation details; ARC Prize leaderboard scores are also computed on a semi-private set (ARC Prize, 2025).

Soft and hard distillations reported in Table 3 under our fixed default setting (w=0.25) both improve over the rollout-KD baseline.

Sudoku parameter scaling. Table 4 illustrates a compute–parameter trade-off: TRM-Planner can recover much of the performance lost when shrinking the student by shifting computation to training time (offline cache generation + distillation), while keeping *student-only* inference unchanged. In our setup, a 5.06M TRM-Planner student (hard KD) outperforms a 7.00M TRM baseline at test time, despite using additional offline/training-time compute to construct the teacher cache. Thus, TRM-Planner trades training-time compute and storage for a smaller inference-time model rather than improving accuracy at equal training cost.

Table 4: Sudoku parameter scaling (student and teacher have the same size). Metrics at $t_{\text{eval}} = T_{\max}$. **TRM is the baseline.**

H	Params (M)	Method	ExactAcc	TokAcc
384	3.55	TRM (reproduced baseline)	0.1236	0.6745
384	3.55	TRM-Planner (hard)	**0.5644**	**0.9035**
448	5.06	TRM (reproduced baseline)	0.6968	0.8907
448	5.06	TRM-Planner (hard)	**0.7696**	**0.9220**
512	7.00	TRM (reproduced baseline)	0.6869	0.8874

We defer post-hoc sensitivity sweeps (over $w, q, T, \sigma, \beta, K$) and cache-diversity audits to Appendix C. In these post-hoc diagnostics, stronger KD/filtering can overshoot (higher Peak but worse fixed-budget Final). Cache diversity via rollout noise is non-monotonic: σ=0.00 collapses to near-duplicate cache entries, moderate noise (e.g., σ=0.02) induces small but nonzero logit diversity and achieves the highest Peak accuracy in our sweep (Table 9), while larger noise (e.g., σ=0.04) increases diversity further but can degrade downstream accuracy. In this case, diversity can help with careful tuning (and/or checkpoint selection) but is not a guaranteed improvement.

9 Conclusion

We introduced TRM-Planner, a two-stage teacher-cache distillation recipe that performs offline target planning and caches teacher supervision for student distillation. Across Sudoku-Extreme and ARC1/ARC2, cached-teacher distillation provides consistent gains over our reproduced TRM baselines without changing inference-time computation. ARC1 controls suggest that cache-based distillation is the primary driver, while the incremental benefit of planning/selection (including diversity) is task-dependent.

Limitations

TRM-Planner improves supervision without changing the TRM architecture, but it introduces an offline generation step that uses training labels to score candidate steps (label-aware target selection). This makes the approach inapplicable to fully unlabeled settings without additional assumptions. Additionally, we do not use the cache or planner at test time; this keeps inference-time computation unchanged but does not explore potential gains from test-time search.

Caching logits can incur nontrivial storage overhead for long sequences and large vocabularies. In this work, we store ARC logits in fp32 to avoid numerical surprises and cache only a small top-K set per key, but scaling to much larger datasets will likely require compression (e.g., bf16/fp16), caching fewer targets (e.g., top-p), and/or caching only supervised positions.

The effectiveness of diversity depends on dataset structure and the strictness of teacher-quality filters. On tasks with essentially unique correct outputs and near-full supervision (e.g., Sudoku), diversity can collapse and cache selection may function primarily as quality improvement rather than providing multiple useful modes.

Ethical Considerations

This work studies training procedures on puzzle-style benchmark datasets (Sudoku and ARC-style tasks) and does not involve human subjects, personal data, or user-generated content.

We follow the licenses and usage terms of all datasets and repositories used, and we document the preprocessing, augmentation, and evaluation code required to reproduce the reported results. The primary risk introduced by TRM-Planner is increased offline computation and storage due to caching teacher outputs; we report these costs and cache sizes alongside accuracy results.

We have used an AI assistant for writing support (e.g., editing for clarity and suggesting phrasing/organization). All technical content, experiments, and reported results were produced and verified by the authors.

References

ARC Prize. 2025. Arc prize testing policy. Website. Accessed 2026-01-03.

Lei Jimmy Ba and Rich Caruana. 2014. Do deep nets really need to be deep? In *Advances in Neural Information Processing Systems*, volume 27. Curran Associates, Inc.

Cristian Bucilua, Rich Caruana, and Alexandru Niculescu-Mizil. 2006. Model compression. In *Proceedings of the 12th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '06, page 535–541, New York, NY, USA. Association for Computing Machinery.

Jaime Carbonell and Jade Goldstein. 1998. The use of mmr, diversity-based reranking for reordering documents and producing summaries. In *Proceedings of the 21st Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR '98, page 335–336, New York, NY, USA. Association for Computing Machinery.

Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Ponde de Oliveira Pinto, Jared Kaplan, Harri Edwards, Yuri Burda, Nicholas Joseph, Greg Brockman, Alex Ray, Raul Puri, Gretchen Krueger, Michael Petrov, Heidy Khlaaf, Girish Sastry, Pamela Mishkin, Brooke Chan, Scott Gray, and 39 others. 2021. Evaluating large language models trained on code. *Preprint*, arXiv:2107.03374.

Mostafa Dehghani, Stephan Gouws, Oriol Vinyals, Jakob Uszkoreit, and Lukasz Kaiser. 2019. Universal transformers. In *International Conference on Learning Representations*.

Tommaso Furlanello, Zachary Lipton, Michael Tschannen, Laurent Itti, and Anima Anandkumar. 2018. Born again neural networks. In *Proceedings of the 35th International Conference on Machine Learning*, volume 80 of *Proceedings of Machine Learning Research*, pages 1607–1616. PMLR.

Marjan Ghazvininejad, Omer Levy, Yinhan Liu, and Luke Zettlemoyer. 2019. Mask-predict: Parallel decoding of conditional masked language models. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, pages 6112–6121, Hong Kong, China. Association for Computational Linguistics.

Alex Graves. 2016. Adaptive computation time for recurrent neural networks. *Preprint*, arXiv:1603.08983.

Jiatao Gu, James Bradbury, Caiming Xiong, Victor O.K. Li, and Richard Socher. 2018. Non-autoregressive neural machine translation. In *International Conference on Learning Representations*.

Geoffrey Hinton, Oriol Vinyals, and Jeff Dean. 2015. Distilling the knowledge in a neural network. *Preprint*, arXiv:1503.02531.

Alexia Jolicoeur-Martineau. 2025. Less is more: Recursive reasoning with tiny networks. *Preprint*, arXiv:2510.04871.

Yoon Kim and Alexander M Rush. 2016. Sequence-level knowledge distillation. In *Proceedings of the 2016 conference on empirical methods in natural language processing*, pages 1317–1327.

Alex Kulesza and Ben Taskar. 2012. Determinantal point processes for machine learning. *Foundations and Trends in Machine Learning*, 5(2–3):123–286.

Jason Lee, Elman Mansimov, and Kyunghyun Cho. 2018. Deterministic non-autoregressive neural sequence modeling by iterative refinement. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pages 1173–1182.

Lucas Prieto, Melih Barsbey, Pedro Mediano, and Tolga Birdal. 2025. Grokking at the edge of numerical stability. In *International Conference on Learning Representations (ICLR)*, pages 81151–81168.

Adriana Romero, Nicolas Ballas, Samira Ebrahimi Kahou, Antoine Chassang, Carlo Gatta, and Yoshua Bengio. 2015. Fitnets: Hints for thin deep nets. In *International Conference on Learning Representations (ICLR)*.

Ashwin Vijayakumar, Michael Cogswell, Ramprasaath Selvaraju, Qing Sun, Stefan Lee, David Crandall, and Dhruv Batra. 2016. Diverse beam search: Decoding diverse solutions from neural sequence models. *Preprint*, arXiv:1610.02424.

Qizhe Xie, Minh-Thang Luong, Eduard Hovy, and Quoc V. Le. 2020. Self-training with noisy student improves imagenet classification. *Preprint*, arXiv:1911.04252.

A Maze reproduction attempts

We attempted to reproduce TRM and TRM-Planner results on Maze-30x30-Hard. In our runs, token accuracy is high while exact match remains low (4–6%). This is expected for long sequences: even a small per-token error rate can drive exact match to very low values. For reference, Jolicoeur-Martineau (2025, Table 4) reports 85.3% on Maze-Hard for TRM-Att (7M). We were unable to reproduce this result in our environment.

Table 5: Maze-30x30-Hard results (test split). We report exact match and token accuracy on supervised tokens. **TRM is the baseline.**

Method	ExactAcc ↑	TokenAcc ↑
TRM (reproduced baseline)	0.0490	**0.9658**
TRM-Planner (soft KD, sample-one, w=0.25)	0.0450	0.9494
TRM-Planner (hard KD, w=0.25)	**0.0580**	0.9603

Overall, while hard KD shows a small improvement in exact match as shown in Table 5, Maze remains unstable in our environment and does not provide a reliable benchmark for our main claims. We therefore treat Maze as a diagnostic dataset and do not include it as a headline result.

B Implementation details

Model. We use the same configuration as in the TRM work: a non-causal 2-layer transformer refinement operator (L_{layers}=2, hidden size 512, 8 heads, SwiGLU expansion 4, RoPE) and a learned per-puzzle embedding (length 16, dimension 512). The ACT cap is $T_{\max}$=16 steps. For ARC and Maze we use H_{cycles}=3, L_{cycles}=4; Sudoku uses L_{cycles}=6.

Teacher/student initialization. In all reported runs, the teacher and student share the same TRM architecture. Offline cache generation uses a frozen checkpoint, and student training initializes from the same checkpoint (`+load_checkpoint=<TEACHER_CKPT>`).

Offline cache generation. Unless otherwise noted, we use $(T_{\max}, K_b, R, K) = (16, 4, 8, 2)$ with $(\alpha, \beta, \lambda) = (1.0, 0.0, 0.01)$, STABLEJS-MMR selection ($\lambda_{\text{sel}} = 0.7$), and store teacher logits in float32. ARC caches use example-level keys with $\sigma = 0.02$; Sudoku caches use puzzle-level keys with $\sigma = 0.02$.

Training. Distillation weight w varies depending on each run, and quality threshold q=0.5 and temperature T=1 for soft KD. ARC runs follow the defaults for optimization and regularization: AdamATan2 with $(\beta_1, \beta_2) = (0.9, 0.95)$, lr= 10^{-4}, 2,000 warmup steps; weight decay= 0.1, puzzle-embedding lr= 10^{-2}, and puzzle-embedding weight decay= 0.1. ARC uses `global_batch_size`= 768, `epochs`= 100,000. In our runs we enable EMA (`ema=True`, `ema_rate`= 0.999).

Sudoku and Maze use the same optimizer family and LR schedule (lr= 10^{-4}, 2,000 warmup, no decay, EMA enabled), but override: puzzle-embedding lr= 10^{-4}, weight decay= 1.0, puzzle-embedding weight decay= 1.0, `epochs`= 50,000. We use `global_batch_size`= 768 for Sudoku and 512 for Maze.

We evaluate predictions every fixed interval and report metrics at $t_{\text{eval}} = T_{\max}$.

Hardware and seeds. ARC1/ARC2/Sudoku use 1 node of 8×H200 GPUs; Maze uses 1 node of 4×L40S GPUs. Unless otherwise noted we use seed 0.

C Sudoku post-hoc sweeps on the test split

Important note (no test-set feedback). The tables in this section report *post-hoc* diagnostics computed on the Sudoku test split (including intermediate-checkpoint maxima and the step at which they occur). **We did not use these diagnostics for model selection, early stopping, or hyperparameter tuning**, and no reported configuration in Table 3 was chosen by comparing test-set curves across settings. The results below are included only to characterize overshooting and sensitivity.

Table 6: Sudoku A1: teacher-weight sweep (soft KD; $q = 0.5$, $T = 1$). We report Best and Final to expose overshooting.

w	test/exact (Peak)	step@Peak	test/exact (Final)	train/exact (Final)
0.25	0.8277	65100	**0.8277**	0.7573
0.75	0.8161	45570	0.7836	0.9184
1.00	**0.8424**	39060	0.7766	0.9212

Table 7: Sudoku A2: quality-threshold sweep (soft KD; $w = 1.0, T = 1$). Aggressive filtering can increase Best but worsen Final due to overshooting.

q	test/exact (Peak)	step@Peak	test/exact (Final)	train/exact (Final)
None	0.7995	32550	0.6766	0.9012
0.7	0.8287	58590	**0.7927**	0.9205
0.3	**0.8458**	39060	0.6398	0.9652

Table 8: Sudoku A3/A4: temperature sweep and hard KD under $(w = 1.0, q = 0.3)$. We report Best and Final. Increasing temperature does not improve the Best achieved at $T = 1$, and several settings overshoot late.

Setting	T	w	test/exact (Peak)	step@Peak	test/exact (Final)	train/exact (Final)
Soft KD (baseline)	1	1.00	**0.8458**	39060	0.6398	0.9652
Soft KD (strength matched)	2	0.25	0.8296	19530	0.7286	0.9335
Soft KD (strength matched)	4	0.0625	0.8358	39060	**0.7692**	0.9344
Hard KD	0	1.00	0.8266	32550	0.5149	0.9674

Table 9: Sudoku Stage B: generation noise sweep (soft KD; $w = 1.0, q = 0.3, T = 1$) with regenerated caches at different noise scales.

σ	test/exact (Peak)	step@Peak	test/exact (Final)	train/exact (Final)
0.00	0.8094	65100	**0.8094**	0.9136
0.02	**0.8458**	39060	0.6398	0.9652
0.04	0.6882	65100	0.6882	0.9745

Table 10: Sudoku Stage C: planner-score halting weight β (soft KD; q=0.5, T=1).

β	w	test/exact (Peak)	step@Peak	test/exact (Final)	train/exact (Final)
0.00	0.25	0.8277	65100	0.8277	0.7573
0.00	1.00	**0.8458**	39060	0.6398	0.9652
0.25	0.25	0.8091	52080	0.8062	0.7263
0.25	1.00	0.8441	32550	0.5873	0.9615
1.00	0.25	0.8133	65100	**0.8133**	0.7500
1.00	1.00	0.8366	45570	0.6666	0.9544

Table 11: Sudoku Stage D: cache size K sweep (soft KD; w=0.25, q=0.5, T=1, σ=0.02).

K	test/exact (Peak)	step@Peak	test/exact (Final)	train/exact (Final)
2	**0.8277**	65100	**0.8277**	0.7573
3	0.7962	65100	0.7962	0.7407

C.1 Sudoku sensitivity summary (post-hoc)

Stagewise sweep summary. Reusing the same teacher cache, increasing w can increase intermediate-checkpoint accuracy (Peak) but is less stable at the end (Table 6): for example, under $(q$=0.5, T=1), w=1.0 achieves a higher best but overshoots compared to the more stable w=0.25. Fixing $(w$=1.0, T=1), aggressive filtering (q=0.3) yields the strongest best but can collapse in Final (see Table 7), consistent with teacher-dominance/overshooting; q=0.7 is substantially more stable. A temperature sweep that keeps wT^2 constant does not improve over T=1 as shown in Table 8, and hard KD can be unstable under aggressive (w, q).

Halting-confidence term. In our post-hoc runs (Table 10), adding a halting-confidence term ($\beta >$ 0) in the planner score did not improve fixed-budget Final accuracy under the tested settings.

Cache size K. We vary the number of cached teacher entries per puzzle (K), holding the rest of the A1 setting fixed (see Table 11). Compared to the default K=2, increasing to K=3 degrades test exact accuracy in this run, suggesting that for Sudoku (near-unique outputs) adding additional cached entries can introduce lower-quality variants.

Diversity via rollout noise. Table 12 shows teacher-cache diversity. On Sudoku, cached entries are often near-duplicates even with STABLEJS-MMR. With σ=0.00, rollouts can collapse so the top-K cached entries become identical on supervised tokens (and masked StableMax-JS is ≈ 0), making selection effectively score-only. Moderate noise (σ=0.02) produces small but nonzero logit diversity and achieves the highest Peak accuracy in our post-hoc sweep (Table 9), although it can overshoot under aggressive KD at a fixed training budget (lower Final). Larger noise (e.g., σ=0.04) increases diversity further but can degrade downstream accuracy. Overall, diversity can help but is sensitive to (w, q) and checkpoint selection; we therefore treat it as a tunable heuristic.

Table 12: Cache audit on 2,000 sampled Sudoku training instances (seed 0). "masked_identical" is the fraction of examples where the top-2 cached entries match exactly on supervised tokens; "StableMax-JS" is the mean masked JS divergence.

Cache setting	masked_identical	StableMax-JS mean
σ=0.00	1.0000	0.00000
σ=0.02	0.8860	0.00506
σ=0.04	0.8705	0.00667

D ARC compute and cache sizes

We present the observed ARC compute and cache sizes in Table 13.

Table 13: Compute and cache storage for ARC runs in our environment (1 node, 8×H200). Cache generation may include retry overhead due to cluster timeouts.

Run	GPU-hours	Cache disk
ARC1 TRM teacher	319.7	–
ARC1 TRM-Planner cache generation (observed)	652.0	424 GB
ARC1 TRM-Planner student (soft, w=1.0)	283.9	–
ARC2 TRM teacher	384.2	–
ARC2 TRM-Planner cache generation (observed)	806.9	602 GB
ARC2 TRM-Planner student (hard, w=0.25)	402.8	–