A. Proof of Theorem 1

Theorem 1. *$\mathcal{P}$ is trace-injective $\iff$ for all trace prefixes $t' = [c_1, \ldots, c_h]$, the set of possible outputs $F(t')$ is partitioned by the next choice $c_{h+1} \sim \mathcal{C}(\pi_{h+1})$, i.e., the set $\{F([c_1, \ldots, c_h, c_{h+1}]) \mid c_{h+1} \in \{0, \ldots, len(\pi_{h+1}) - 1\}\}$ is a partition of $F(t')$.*

Proof. For any trace prefix $[c_1, \ldots, c_h]$, define the collection

$$\mathcal{F}([c_1, \ldots, c_h]) = \{F([c_1, \ldots, c_h, c_{h+1}]) \\ \mid c_{h+1} \in \{0, \ldots, \text{len}(\pi_{h+1}) - 1\}\}.$$

First, if $\mathcal{P}$ is trace-injective, then for any trace prefix $[c_1, \ldots, c_h]$, and any $c \neq c'$, let $y_1 \in F([c_1, \ldots, c_h, c])$ and $y_2 \in F([c_1, \ldots, c_h, c'])$. Then there exist traces

$$t^{(1)} = [c_1, \ldots, c_h, c_{h+1}^{(1)}, \ldots, c_{h_1}^{(1)}]$$

and

$$t^{(2)} = [c_1, \ldots, c_h, c_{h+1}^{(2)}, \ldots, c_{h_2}^{(2)}]$$

such that

(a) $c_{h+1}^{(1)} = c$ and $c_{h+1}^{(2)} = c'$,

(b) $f(t^{(1)}) = y_1$ and $f(t^{(2)}) = y_2$.

Clearly $t^{(1)} \neq t^{(2)}$, so because f is injective, we have that $y_1 \neq y_2$. Also $\cup_{c'} F([c_1, \ldots, c_h, c']) = F([c_1, \ldots, c_h])$. This means that $\mathcal{F}([c_1, \ldots, c_h])$ is a partition of $F([c_1, \ldots, c_h])$.

Conversely, assume that $\mathcal{F}(t')$ is a partition of $F(t')$ for any trace prefix t'. Let

$$t^{(1)} = [c_1^{(1)}, \ldots, c_{h_1}^{(1)}]$$

and

$$t^{(2)} = [c_1^{(2)}, \ldots, c_{h_2}^{(2)}]$$

be distinct traces. Let h be the length of their longest common prefix, so $c_i^{(1)} = c_i^{(2)}$ for all $1 \leq i \leq h$, and $c_{h+1}^{(1)} \neq c_{h+1}^{(2)}$. By definition,

$$f(t^{(1)}) \in F([c_1^{(1)}, \ldots, c_h^{(1)}, c_{h+1}^{(1)}])$$

and

$$f(t^{(2)}) \in F([c_1^{(2)}, \ldots, c_h^{(2)}, c_{h+1}^{(2)}]) \\ = F([c_1^{(1)}, \ldots, c_h^{(1)}, c_{h+1}^{(2)}]).$$

But these two sets are disjoint, because $\mathcal{F}([c_1^{(1)}, \ldots, c_h^{(1)}])$ partitions the set $F([c_1^{(1)}, \ldots, c_h^{(1)}])$. Therefore, $f(t^{(1)}) \neq f(t^{(2)})$, establishing that f is injective and that $\mathcal{P}$ is trace-injective. □

B. Proof of Correctness

Theorem 2. *Let $\mathcal{P}$ be a discrete randomized program that terminates, and let $P(t)$ be the probability that $\mathcal{P}$ runs with trace t. Suppose we have already sampled distinct traces $t_1, \ldots, t_j$. If, at any* UniqueRandomizer *trie node n we move to a child c with probability proportional to $\text{mass}(c)$, then upon reaching a leaf node, the resulting trace is drawn from $P(t \mid t \notin \{t_1, \ldots, t_j\})$.*

Proof. Let $n_0, \ldots, n_h$ be any root-to-leaf path, where n_0 is the root and n_h is the leaf. Let t be the trace corresponding to n_h. According to Equation (3),

$$\text{mass}(n_h) = \begin{cases} 0 & \text{if } t \in \{t_1, \ldots, t_j\} \\ P(t) & \text{otherwise.} \end{cases}$$

We complete the proof by showing that the leaf n_h is reached with the desired probability:

$$\begin{aligned} & P(n_h \text{ is reached}) \\ &= \prod_{i=1}^{h} P(n_i \text{ is the selected child of } n_{i-1}) \\ &= \prod_{i=1}^{h} \frac{\text{mass}(n_i)}{\sum_{c \in \text{children}(n_{i-1})} \text{mass}(c)} \\ &= \prod_{i=1}^{h} \frac{\text{mass}(n_i)}{\text{mass}(n_{i-1})} \qquad (7) \\ &= \frac{\text{mass}(n_h)}{\text{mass}(n_0)} \\ &= \frac{\text{mass}(n_h)}{1 - \sum_{i=1}^{j} P(t_i)} \\ &= \begin{cases} 0 & \text{if } t \in \{t_1, \ldots, t_j\} \\ \frac{1}{1 - \sum_{i=1}^{j} P(t_i)} P(t) & \text{otherwise} \end{cases} \\ &= P(t \mid t \notin \{t_1, \ldots, t_j\}). \end{aligned}$$

Equality (7) holds because a non-leaf node's mass equals the sum of its children's mass values. □

C. Detecting Exhausted Nodes

We say that a trie node is *exhausted* if it has zero unsampled probability mass, i.e., all of its probability mass is sampled. Due to floating-point errors, a node's mass might not be set to exactly zero after it should be exhausted. We handle this by carefully propagating the information that a given node has zero unsampled probability mass.

When a node n is marked as a leaf, we directly assign $\text{mass}(n) := 0$. Then, when subtracting mass from one of n's ancestors a, we first check if $\text{mass}(c) = 0$ for all children c of a. If so, we directly set $\text{mass}(a) := 0$ instead of using a subtraction operation. With this approach, a node's mass will be exactly 0 after all of its descendent leaves are sampled. Algorithm 3 includes this process, elaborating on the pseudocode in Algorithm 2.

D. Locally Modifying the Factorized Probability Distribution

A slight modification of *UniqueRandomizer*'s trie allows for efficient local updates to the factorized probability distribution. Instead of storing unsampled probability masses of nodes, the modified trie nodes now store the *unsampled fraction* of the node's total probability mass. Edges in the trie now store the initial probability of following that edge from the source node, as given in the probability distribution provided by $\mathcal{P}$.

Note that the unsampled probability mass of a node n is equal to the product of the edge probabilities from the root to n, times the unsampled fraction at n. Therefore, by accumulating the product of edge probabilities while walking down the trie, we can compute the unsampled probability mass of nodes, so we can recreate the original *UniqueRandomizer* behavior with the modified trie.

This decomposition enables local modifications to the factorized probability distribution. More precisely, suppose that a trie node n has k children, denoted $n_1, \ldots, n_k$, and n initially has outward edge probabilities of $p_1, \ldots, p_k$. We wish to change these edge probabilities to $p'_1, \ldots, p'_k$, so that further samples come from the updated probability distribution and previously-seen samples are still avoided. We do this by updating the trie in the following way. First, we directly replace n's outward edge probabilities with the desired $p'_1, \ldots, p'_k$. Then, we compute the new unsampled fraction at n with a weighted average of n's children:

$$\mathrm{unsampledFraction}(n) := \sum_{i=1}^{k} \mathrm{edgeProbability}(n, n_i) \cdot \mathrm{unsampledFraction}(n_i).$$

Finally, we perform a similar update for all of n's ancestors in upward order (with the root being updated last). After these updates, all values in the trie reflect the new probability distribution.

E. Program Synthesis Experiment Details

For the program synthesis task, we train a Transformer model (Vaswani et al., 2017) to translate lines of pseudocode to lines of C++ code. We use the Transformer implementation in the Trax framework[6]. The Transformer uses 2 attention heads, 3 hidden layers, a filter size of 1024, and a hidden dimension size of 512. We train using ADAM with learning rate 0.05 and batch size 512 for 12,000 steps, which is approximately when the models achieve their lowest evaluation loss. We use linear learning rate warmup for the first 1,000 steps. These hyperparameters were chosen from the search space in Table 3, selecting the run with the lowest evaluation loss at the end of training. As in Kulal et al. (2019), we withhold 10% of the training examples as the validation set.

Some of the shorter lines of code in the SPoC dataset have no pseudocode. In some of these instances, we augment the line with pseudocode ourselves. Specifically, if the line is exactly "`}`" or "`};`" we provide pseudocode "end", if the line is exactly "`int main() {`" we provide pseudocode "main", and if the line is exactly "`return 0;`" we provide pseudocode "return".

[6] `https://github.com/google/trax`.

Table 3: The search space used for tuning the Transformer model.

Hyperparameter	Search space	Selected value
Learning rate	{0.05, 0.075, 0.1, 0.15}	0.05
Hidden layers	{1, 2, 3}	3
Hidden dimension size	{512, 1024}	512
Attention heads	{2, 4}	2
Filter size	{512, 1024}	1024

Algorithm 3 *UniqueRandomizer*, with careful detection of exhausted nodes

```
▷ Called once to initialize the data structure
procedure INITIALIZE()
    root ← TRIENODE(parent = ∅, mass = 1)
    cur ← root

▷ Whether node is completely sampled
procedure EXHAUSTED(node)
    if node is a leaf then
        return True
    if node has never been sampled from before then
        return False
    return whether all of node's children have 0 mass

▷ Called when 𝒫 requests a random choice
procedure RANDOMCHOICE(π)
    if EXHAUSTED(cur) then
        raise Error("no more unique traces exist")
    if cur's children are not initialized yet then
        for 0 ≤ i < len(π) do
            cur.children[i] ← TRIENODE(
                parent = cur, mass = π[i] · cur.mass)
    index ← randomly sample i with probability
            ∝ cur.children[i].mass
    cur ← cur.children[index]
    return index

▷ Called after 𝒫 terminates
procedure PROCESSTERMINATION()
    mark cur as a leaf
    node ← cur
    while node ≠ ∅ do
        if EXHAUSTED(node) then
            node.mass ← 0
        else
            node.mass ← max{node.mass − cur.mass,
                            0}
        node ← node.parent
    cur ← root
```