

A Additional Proofs

For any \mathbb{D} , let $d = \lceil \log(|\mathbb{D}|) \rceil + 1$ and let $\Psi : \mathbb{D} \rightarrow \{0, 1\}^d$ be a one-to-one mapping of tokens to Boolean vectors, s.t. $\Psi_1(s) = 1$ for all $s \in \mathbb{D}$.

Definition 9. A function $f : \mathbb{D}^L \rightarrow \mathbb{D}$ is called k -Junta if there exists a set of separate indexes $i_1, \dots, i_k \in [L]$ and function $g : \mathbb{D}^k \rightarrow \mathbb{D}$ s.t. $f(\mathbf{x}) = g(x_{i_1}, \dots, x_{i_k})$.

Lemma 10. For every k -Junta $f : \mathbb{D}^L \rightarrow \mathbb{D}$, there exists a tree \mathcal{T} of size $O(|\mathbb{D}|^k)$ and depth $O(k \log |\mathbb{D}|)$ s.t. $\mathcal{T}(\Psi(\mathbf{x})) = f(\mathbf{x})$ for all $\mathbf{x} \in \mathbb{D}^L$.

Proof. Let \mathcal{T} the perfect binary tree of depth dk , where each level of the tree corresponds to a pair $(j, l) \in [k] \times [d]$, and all the nodes at the level implement the condition $\Psi_l(x_{i_j}) \geq 1$. Observe that in this construction, each leaf correspond to a specific choice of values for $\Psi(x_{i_1}), \dots, \Psi(x_{i_k})$, and we can set its output to be $g(x_{i_1}, \dots, x_{i_k})$. \square

Proof of Theorem 3 Let \mathcal{A} be some automaton, defined by transition function $\delta : Q \times \Sigma \rightarrow Q$, and we can arbitrarily extend it to $\delta : \mathbb{D}^2 \rightarrow \mathbb{D}$ s.t. $\delta(x, \langle \text{PAD} \rangle) = q_0$ for all $x \in \mathbb{D}$. Then, from Theorem 10 there exists some tree \mathcal{T} of size $O(|\mathbb{D}|^2)$ s.t. for all $\mathbf{x} \in \mathbb{D}^L$ it holds that $\mathcal{T}(\Psi(\mathbf{x})) = \delta(x_L, x_{L-n})$.

We prove by induction that for all $i \in [n]$ it holds that $\mathcal{T}_i^{\text{AR}}(\mathbf{x}) = q_i$, where q_i is the state of the automaton \mathcal{A} at iteration i .

- Let $\mathbf{z} \in \mathbb{R}^{L,d}$ be the padded output of $\Psi(\mathbf{x})$, i.e. $\mathbf{z} = [\Psi(\langle \text{PAD} \rangle), \dots, \Psi(\langle \text{PAD} \rangle), \Psi(x_1), \dots, \Psi(x_n)]$. Note that since $x_{L-n} = \langle \text{PAD} \rangle$ we have $\mathcal{T}_1^{\text{AR}}(\mathbf{x}) = \mathcal{T}(\mathbf{z}) = \delta(x_L, \langle \text{PAD} \rangle) = q_1$.
- Assume that $\mathcal{T}_{1:i-1}^{\text{AR}}(\mathbf{x}) = (q_1, \dots, q_{i-1})$. Therefore,

$$\begin{aligned} \mathcal{T}_i^{\text{AR}}(\mathbf{x}) &= \mathcal{T}(\Psi(\langle \text{PAD} \rangle), \dots, \langle \text{PAD} \rangle, x_1, \dots, x_n, q_1, \dots, q_{i-1}) \\ &= \delta(q_{i-1}, x_i) = q_i \end{aligned}$$

Therefore, the required follows. \square

Proof of Theorem 6 We encode the state of the Turing machine by a string $\mathbf{s} \in \mathbb{D}^{M+1}$ as follows: if the head is in state $q \in Q$ and at position $i \in [M]$, and the memory is $m_1, \dots, m_M \in \Sigma$, we set $\mathbf{s} = (m_1, \dots, m_{i-1}, q, m_i, \dots, m_M)$. That is, we add a token indicating the state of the head before the cell where the head is located. Let $\delta : Q \times \Sigma \rightarrow Q \times \Sigma \times \{\langle \text{LEFT} \rangle, \langle \text{RIGHT} \rangle\}$ be the transition function of the Turing machine. We define the following function $g : \mathbb{D}^4 \rightarrow \mathbb{D}^4$:

$$g(\mathbf{s}) = \begin{cases} x_2 & \text{if } x_1, x_2, x_3 \notin Q \\ q & \text{if } x_1 \in Q \text{ and } \delta(x_1, x_2) = (q, \alpha, \langle \text{RIGHT} \rangle) \\ \alpha & \text{if } x_1 \in Q \text{ and } \delta(x_1, x_2) = (q, \alpha, \langle \text{LEFT} \rangle) \\ \alpha & \text{if } x_2 \in Q \text{ and } \delta(x_2, x_3) = (q, \alpha, \langle \text{RIGHT} \rangle) \\ x_1 & \text{if } x_2 \in Q \text{ and } \delta(x_2, x_3) = (q, \alpha, \langle \text{LEFT} \rangle) \\ x_2 & \text{if } x_3 \in Q \text{ and } \delta(x_3, x_4) = (q, \alpha, \langle \text{RIGHT} \rangle) \\ q & \text{if } x_3 \in Q \text{ and } \delta(x_3, x_4) = (q, \alpha, \langle \text{LEFT} \rangle) \end{cases}$$

Observe that the function $f : \mathbb{D}^{M+1} \rightarrow \mathbb{D}^{M+1}$ s.t. $f_i(\mathbf{s}) = g(s_{i-1}, s_i, s_{i+1}, s_{i+2})$ exactly defines the transition between the encoded states of the Turing machine. Namely, if the state of the machine at iteration i is \mathbf{s} , then the state at iteration $i + 1$ is $f(\mathbf{s})$. We slightly modify g to handle the generation of the first iteration, as follows:

$$\tilde{g}(\mathbf{s}) = \begin{cases} \langle \text{SEP} \rangle & x_1 = \langle \text{PAD} \rangle \text{ and } x_2 = \langle \text{PAD} \rangle \text{ and } x_3 = \langle \text{PAD} \rangle \\ q_0 & x_1 = \langle \text{PAD} \rangle \text{ and } x_2 = \langle \text{PAD} \rangle \text{ and } x_3 \neq \langle \text{PAD} \rangle \\ \langle \text{SEP} \rangle & x_2 = \langle \text{SEP} \rangle \\ g(\mathbf{s}) & \text{otherwise} \end{cases}$$

Now, from Lemma 10 there exists a tree \mathcal{T} of size $O(|\mathbb{D}|^4)$ s.t. $\mathcal{T}(\Psi(\mathbf{x})) = \tilde{g}(x_1, x_2, x_3, x_4)$.
Let $\mathbf{s}_1, \dots, \mathbf{s}_T \in \mathbb{D}^{M+1}$ the encodings of the state of the Turing machine at iterations $1, \dots, T$. Let $\mathbf{x} \in \mathbb{D}^L$ be the encoding of the input, starting with $\langle \text{PAD} \rangle$ tokens, followed by one $\langle \text{BOS} \rangle$ token and the input string. Denote the output of the ARDT \mathcal{T}^{AR} after $T \cdot (M + 2)$ given the input \mathbf{x} , where we split the output into chunks of size $M + 2$ by:

$$\mathcal{T}^{\text{AR}}(\mathbf{x}) = (\mathbf{z}_1, \dots, \mathbf{z}_T) \in \mathbb{D}^{T \cdot (M+2)}, \mathbf{z}_i \in \mathbb{D}^{M+2}$$

Claim: For all $i \in [T]$, it holds that $\mathbf{z}_i = (\langle \text{SEP} \rangle, \mathbf{s}_i)$.

Prove: We prove by induction on i .

- For $i = 1$, notice that the input begins with 3 $\langle \text{PAD} \rangle$ tokens, followed by the input tokens x_1, \dots, x_M , and therefore by definition of \tilde{g} we get $\mathbf{z}_1 = (\langle \text{SEP} \rangle, q_0, x_1, \dots, x_M) = (\langle \text{SEP} \rangle, \mathbf{s}_1)$.
- Assume the required holds for i . First, observe that

$$\mathbf{z}_{i+1,1} = \mathcal{T}(\Psi(s_{i-1,M+1}, \langle \text{SEP} \rangle, s_{i,1}, \dots, s_{i,M+1})) = \langle \text{SEP} \rangle$$

Now, assume that $\mathbf{z}_{i+1,1:j} = (\langle \text{SEP} \rangle, s_{i+1,1}, \dots, s_{i+1,j-1})$. Therefore

$$\begin{aligned} \mathbf{z}_{i+1,j+1} &= \mathcal{T}(\Psi(s_{i,j-1}, s_{i,j}, s_{i,j+1}, \dots, s_{i,M+1}, \langle \text{SEP} \rangle, s_{i+1,1}, \dots, s_{i+1,j-1})) \\ &= g(s_{i,j-1}, s_{i,j}, s_{i,j+1}, s_{i,j+2}) = s_{i+1,j} \end{aligned}$$

and by induction we get $\mathbf{z}_{i+1} = (\langle \text{SEP} \rangle, \mathbf{s}_{i+1})$

Therefore, \mathcal{T} outputs the final token of iteration T after $T(M + 2)$ steps of auto-regression, which proves the theorem. \square

B Additional Implementation Details

B.1 Hardware & Computational Cost

Our experiments were conducted on a single NVIDIA A100 GPU. For the Tiny Stories experiments, the training process took approximately 1 hour, and it required about 1 second to generate 20 words during the inference phase.

B.2 Dataset Details

Tiny Stories. As shown in Tab. 3, the training and validation datasets of Tiny Stories contain 147,273 and 21,990 stories, respectively. We use NLTK Bird et al. (2009) as the tokenizer to obtain 420,351,665 and 4,329,963 tokens from the training dataset. In the training dataset and validation dataset, the number of words in the vocabulary is 27,455 and 11,273, respectively.

BIG-Bench-Hard is a dataset contains the selection of 23 difficult tasks from the BIG-Bench. These tasks are identified by their resistance to being outperformed by prior language model evaluations when compared to the average human evaluator. The BIG-Bench-Hard tasks often demand complex, multi-step reasoning, and the use of few-shot prompting without CoT, as previously utilized in BIG-Bench evaluations Srivastava et al. (2023), significantly underrepresents the true potential and performance of language models.

Four representative reasoning tasks we select for evaluate our ARDTs:

- (1) *Boolean Expressions*. Example: not (True) and (True). Answer: False.
- (2) *Navigate*. Example: If you follow these instructions, will you return to the starting point? Instructions: Turn left. Take 5 steps. Turn right. Answer: No.
- (3) *Web-of-Lies*. Example: Delbert tells the truth. Delfina says Delbert lies. Antwan says Delfina tells the truth. Does Delfina tell the truth? Answer: No.
- (4) *Sports Understanding*. Example: Is the following sentence plausible? "Elias Lindholm beat the buzzer." Answer: No.

Table 3: Basic Information about the Tinstories Dataset.

	Training dataset	Validation dataset
The number of stories	147,273	21,990
The number of tokens	420,351,665	4,329,963
The word count of each story.	54 - 5,498	63 - 4,254
Vocabulary	27455	11274

512 B.3 Details about the Visualization of the Decision Trees

513 To enable visualization that treats words as features, as shown in Algorithm 1, we map word
514 embeddings into a lower-dimensional space. This process utilizes three primary inputs: word
515 embeddings W in an $N \times 100$ matrix, where N represents the number of words and 100 the
516 dimensionality of each embedding; cluster centers C in a 20×100 matrix, indicating 20 clusters
517 within the 100-dimensional embedding space; and a mapping matrix M sized 100×20 , designed
518 to reduce the embeddings’ dimensionality to 20. The algorithm begins with an orthogonalization
519 procedure, applying QR decomposition to the transpose of C (C^T) and returning the first 20 columns
520 of Q^T , thereby establishing an orthogonal basis for the cluster space. It then projects the word
521 embeddings W into this lower-dimensional space by multiplying them with the mapping matrix M .
522 By iterating over each word embedding in W , the algorithm applies this projection and ultimately
523 returns a set of transformed embeddings $\{E_1, \dots, E_N\}$, where each E_i provides a lower-dimensional
524 representation of the corresponding word embedding. This approach allows us to treat each vector
525 value as individual words, facilitating a more intuitive understanding of the data.

Algorithm 1 Map Word Embeddings to Lower Dimensional Space

```

input
  Word Embeddings  $W \subseteq \mathbb{R}^{N \times 100}$ 
  Cluster Centers  $C \subseteq \mathbb{R}^{20 \times 100}$ 
  Mapping Matrix  $M \subseteq \mathbb{R}^{100 \times 20}$ 
procedure ORTHOGONALIZE( $C$ )
   $Q, R \leftarrow \text{QR Decomposition}(C^T)$ 
  return  $Q[:, : 20].T$ 
end procedure

procedure MAPEMBEDDINGS( $W, M$ )
  return  $W \cdot M$ 
end procedure

 $M \leftarrow \text{ORTHOGONALIZE}(C)$ 
for each word embedding  $w_i$  in  $W$  do
   $E_i \leftarrow \text{MAPEMBEDDINGS}(w_i, M)$ 
end for
Return  $\{E_1, \dots, E_N\}$ 

```

526 B.4 The Story Beginnings Used in Our Evaluation

527 In Table 6, we provide 10 examples over 100 story beginnings we used for our experiments on Tiny
528 Stories.

529 B.5 An Example on Multidimensional Scores

530 As an example, we took the story that appears in the example above and cut it after the word "loud".
531 We generated a completion of stories using our ARDTs and baselines, then we prompted GPT-4 to
532 first provide a verbal assessment and then grade the completion. The transcript of interaction is as
533 follows, with the parts produced by GPT-4 in italic font:

Table 4: Generated Chain-of-Thought data for boolean expression.

Input Prompt	not not (not (False)) is	True and False and not True and True is
Generated Prompt	Let's think step by step. Remember that (i) expressions inside brackets are always evaluated first and that (ii) the order of operations from highest priority to lowest priority is "not", "and", "or", respectively. We first simplify this expression "Z" as follows: "Z = not not (not (False)) = not not (A)" where "A = not (False)". Let's evaluate A: A = not (False) = not False = True. Plugging in A, we get: Z = not not (A) = not not (True) = not not False = True. So the answer is True.	Let's think step by step. Remember that (i) expressions inside brackets are always evaluated first and that (ii) the order of operations from highest priority to lowest priority is "not", "and", "or", respectively. We first simplify this expression "Z" as follows: "Z = True and False and not True and True = A and B" where "A = True and False" and "B = not True and True". Let's evaluate A: A = True and False = False. Let's evaluate B: B = not True and True = not (True and True) = not (True) = False. Plugging in A and B, we get: Z = A and B = False and False = False. So the answer is False.

Table 5: Generated Chain-of-Thought data for navigation.

Input Prompt	If you follow these instructions, do you return to the starting point? Always face forward. Take 2 steps right. Take 9 steps left. Take 7 steps right.	If you follow these instructions, do you return to the starting point? Turn around. Take 1 step. Take 6 steps. Turn around. Take 6 steps. Take 9 steps. Take 1 step.
Generated Prompt	Let's think step by step. We start at the origin (0, 0), facing the positive y-axis. (1) Always face forward: (0, 0), facing the positive y-axis. (2) Take 2 steps right: (0, 2), facing the positive y-axis. (3) Take 9 steps left: (0, -7), facing the positive y-axis. (4) Take 7 steps right: (0, 7), facing the positive y-axis. Since (0, 0) is (0, 0), we are indeed where we started. So the answer is Yes.	Let's think step by step. We start at the origin (0, 0), facing the positive y-axis. (1) Turn around: (0, 0), facing the negative y-axis. (2) Take 1 step: (0, -1), facing the negative y-axis. (3) Take 6 steps: (0, -7), facing the negative y-axis. (4) Turn around: (0, -7), facing the positive y-axis. (5) Take 6 steps: (0, -1), facing the positive y-axis. (6) Take 9 steps: (0, 8), facing the positive y-axis. (7) Take 1 step: (0, 9), facing the positive y-axis. Since (0, 9) is not (0, 0), we are not where we started. So the answer is No.

Table 6: Examples of the story beginnings generated by GPT-4. In the inference phase, we fed these story beginnings generated by GPT-4 into the ARDTs to continue writing the stories.

The beginning of the stories (GPT-4 generated)	
1	Kitty found a bright, lost treasure.
2	Thirsty, they sought a hidden pond.
3	Roxy climbed, seeking icy hill's peak.
4	Billy's idea sparked an unlikely friendship.
5	Leaves underfoot, Roxy ascended triumphantly.
6	Daisy, small yet happy, met Max.
7	Max's playful antics delighted tiny Daisy.
8	Sue's thoughtful gesture warmed her mother.
9	A candle's flame united mother, daughter.
10	The sad cow wished for companionship.

the following exercise, the student is given a beginning of a story. The student needs to complete it into a full story. The exercise tests the student's language abilities and creativity. The symbol * * * marks the separator between the prescribed beginning and the student's completion:

Once upon a time, * * * there was a little girl named Lily. She loved to play outside in the sunshine. One day, she saw a big tree in the garden. She was so excited about it.

Please provide your general assessment about the part written by the student (the one after the * * * symbol). Is it grammatically correct? Is it consistent with the beginning of the story? Pay special attention to whether the student manages to complete the sentence which is split in the middle by the separator * * *.

This is a charming and whimsical story about a little girl named Lily who feels excited upon discovering a big tree in the garden.

The grammar is generally correct.

Overall, the student's completion of the story demonstrates adequate language abilities and creativity. However, adding more details about Lily's exploration, as well as her thoughts and actions after discovering the tree, could make the story more engaging and vivid.

Now, grade the student's completion in terms of grammar, creativity, consistency with the story's beginning and whether the plot makes sense. Moreover, please provide your best guess of what the age of the student might be, as reflected from the completion. Choose from possible age groups: A: 3 or under. B: 4-5. C: 6-7. D: 8-9. E: 10-12. F: 13-16.

Grammar: 9/10, Creativity: 4/10, Consistency: 7/10, Age group: E (10-12)

C Interpretability

One advantage of using decision trees over complex neural networks is the Interpretability of the computational process. Unlike neural networks, which rely on abstract vector operations in high-dimensional space, decision trees implement relatively simple logic, computing their output based on splitting criteria that may be easily displayed to, and interpreted by, humans. That said, recall that our decision trees operate on aggregations of word vector embeddings, which make interpretability harder to achieve. Specifically, each splitting rule of the decision tree is based on the value of a single coordinate, which does not necessarily have an interpretable semantic value when using rotation-invariant word embedding methods such as Word2Vec.

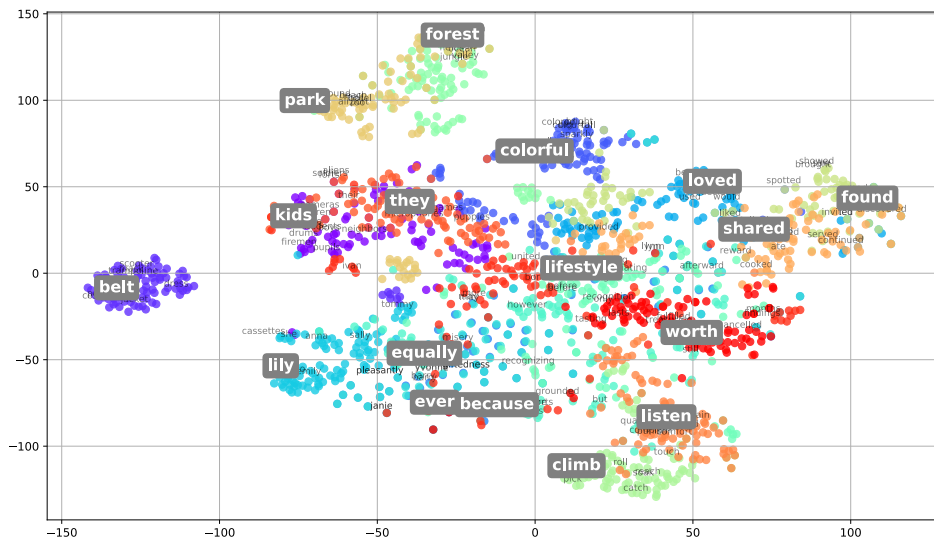


Figure 3: t-SNE [van der Maaten \(2013\)](#) visualization of 20 cluster centers. We selected 20 cluster centers and display 4 words closest to the cluster centers.

In order to generate decision trees with meaningful splitting rules, we modify the word embedding such that single coordinates have specific semantic values. To achieve this, we begin by clustering all the word vectors from the dataset (over 16K words) into 20 clusters using K-means. We then choose one representative word for each cluster, by taking the word that is closest to the center of

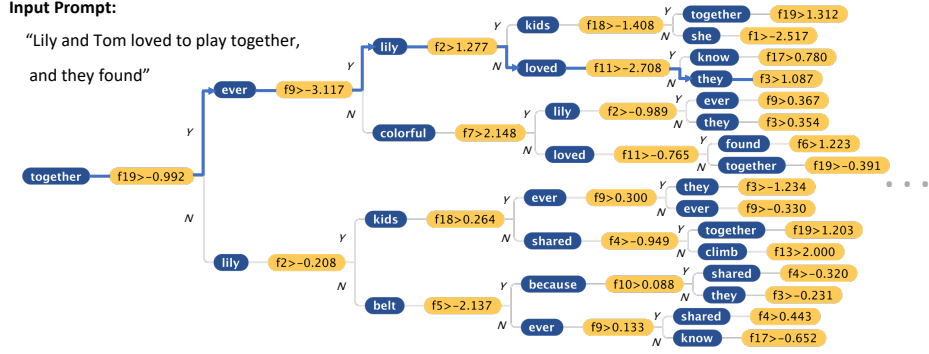


Figure 4: **Track the decision-making process within the decision trees.** We use 'Lily and Tom loved to play together, and they found' as an the input prompt and generate the next word using our ARDTs. We visualize part of the process within the decision tree. Specifically, we visualized 31 nodes of the first decision tree.

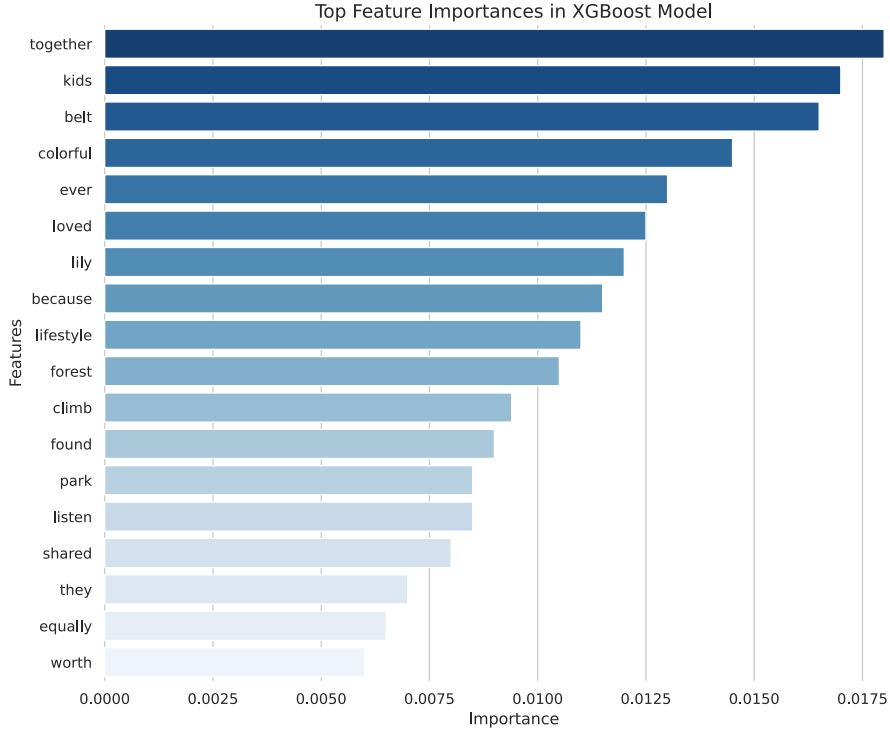


Figure 5: Feature Importance. We present the feature importance of the top 20 words most closely associated with each cluster, based on their average gain.

the cluster in the embedding space (see Figure 3 for an illustration of the clusters and representative words). Now, these words (represented as vectors) form a basis for a *new* 20-dimensional embedding space, which is a linear subspace of the original 100-dimensional space of Word2Vec. We use these basis words to compute the new word embedding, by projecting each vector from the original space into this subspace, and representing the projection as a linear combination of the basis words. Mathematically, if x_1, \dots, x_k are the basis words, we define our new embedding Φ into \mathbb{R}^k by: $\Phi(x) = \arg \min_{z \in \mathbb{R}^k} \|\sum_i z_i \Psi(x_i) - \Psi(x)\|_2$. Observe that each basis word x_i is mapped by Φ to a unit vector e_i . Intuitively, the i -th coordinate of the embedding Φ now represents words that are semantically similar to the word x_i . Now, splitting rules based on the coordinate i can be interpreted as "testing" whether a word similar to x_i appears in the sentence.

578 We visualize one of the decision trees trained on the Tiny Stories Dataset using the new “interpretable”
 579 embedding Φ in Figure 1. Note that, unlike complex neural network architectures, which carry out
 580 opaque computations, the decision process of the ARDT with the new embedding appears to be
 581 semantically meaningful. For example, observe that the word *Lily* appears for three times as the
 582 most relevant word during node splits. Considering *Lily* is a frequently occurring name in the Tiny
 583 Stories dataset, it’s frequent appearance in the tree can be deemed reasonable. We further analyze
 584 the importance of different features by plotting their importance score. We plot the importance of
 585 each cluster, represented by a single word, in Figure 5. We assess the importance of each cluster by
 586 calculating its average gain during every split within the model.

587 In Figure 4, we use the input sentence “Lily and Tom loved to play together and they found” as an
 588 example to visualize part of the decision-making process of the first decision tree in the ensemble.
 589 We note that each feature corresponds to a single cluster, represented by a single word, e.g. the
 590 feature f_2 corresponds to the word “Lily”. That is, the word “Lily” will be mapped to the unit vector
 591 $e_2 = (0, 1, 0, \dots, 0)$. Note that most words (besides the 20 words used as a basis for the embedding),
 592 will be mapped to a linear combination of the basis words, and so can also affect (positively or
 593 negatively) the value of the feature f_2 . Since the input vector is a weighted-average of the embedding
 594 of all words, the decision when splitting on the feature f_2 may be affected by multiple words in the
 595 sentence.