

458 **A Appendix**

459 **A.1 Supplemental Results**

Fig. 6 illustrates model predictions across every Number Game concept in [33].

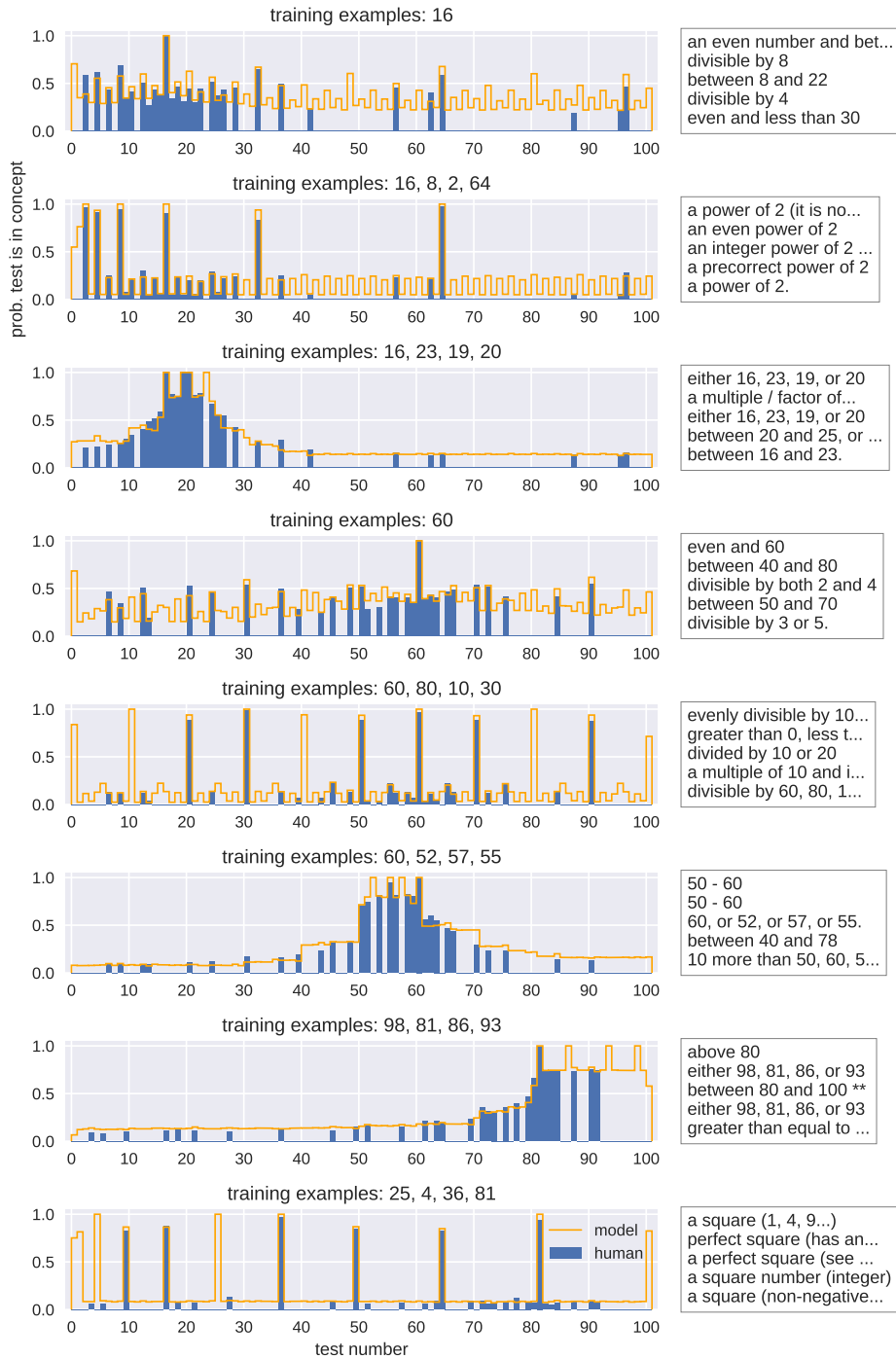


Figure 6: Model predictions across every Number Game concept in [33]

461 Recall that we deduplicated the proposals instead of performing actual importance sampling. Fig. 7
 462 contrasts model fit for importance sampling and deduplication. We originally did deduplication
 463 simply because importance sampling is not possible with GPT-4, and GPT-4 proved necessary for
 464 the logical concepts. On number concepts we used code-davinci-002, from which we can construct
 465 an importance sampler because it exposes the log probability of its samples. On number concepts
 466 deduplication provides a fit that is on-par (actually slightly better) compared to importance sampling
 (Fig. 7).

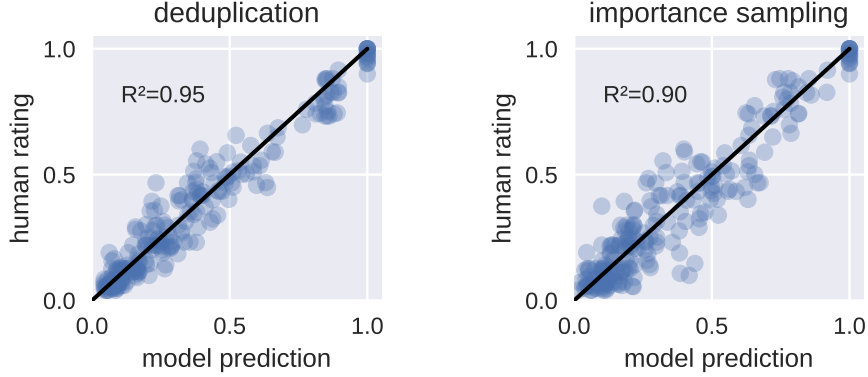


Figure 7: Monte Carlo inference using deduplication instead of importance sampling does not harm model fit to human data. The above figures show Number Game models using a learned prior and 100 samples, and show predictions only on holdout data.

467

468 A.2 Human Study

469 16 participants were recruited primarily through a Slack message sent to a channel populated by
 470 members of our academic department. Participants had an average age 28.1 (stddev 13.6, all over
 471 18), and were 7 male/5 female/1 nonbinary/3 declined to answer. Participants were randomly split
 472 between the concepts of *most common color / least common color*. Each participant went through 15
 473 trials, and took an average of 294s to complete those 15 trials. In exchange for participating in the
 474 study, participants received \$10 in Amazon gift cards. Fig. 8 illustrates the web interface shown to
 475 our human participants, including the cover story.

476 A.3 Modeling

477 A.3.1 Temperature and Platt Transform

478 Adding a temperature parameter T to a model corresponds to computing the posterior via

$$\begin{aligned}
 p_{\text{Temp}}(X_{\text{test}} \in C | X_{1:K}) &\approx \sum_{C \in \{C^{(1)}, \dots, C^{(S)}\}} w^{(C)} \mathbb{1}[X_{\text{test}} \in C], \text{ where} \\
 w^{(C)} &= \frac{(\tilde{w}^{(C)})^{1/T}}{\sum_{C'} (\tilde{w}^{(C')})^{1/T}} \text{ and } \tilde{w}^{(C)} = p(C)p(X_{1:K}|C) \mathbb{1}[C \in \{C^{(1)}, \dots, C^{(S)}\}]
 \end{aligned}
 \tag{10}$$

479 Adjusting the predictions of a model using a Platt transform corresponds to introducing parameters a
 480 and b which transform the predictions according to

$$p_{\text{Platt}}(X_{\text{test}} \in C | X_{1:K}) = \text{Logistic}(b + a \times \text{Logistic}^{-1}(p(X_{\text{test}} \in C | X_{1:K}))) \tag{11}$$

481 For the number game, every model has its outputs transformed by a learned Platt transform. This
 482 is because we are modeling human ratings instead of human responses. We expect that the ratings
 483 correspond to some monotonic transformation of the human’s subjective probability estimates, and so
 484 this transformation gives some extra flexibility by inferring the correspondence between probabilities
 485 and ratings. Logical concept models do not use Platt transforms.

Trial 1: Please read these instructions carefully

You are going to attempt to learn the meaning of a new word in an alien language, which the aliens call "Wudsy." On each trial, you are going to see a collection of shapes at the bottom of the webpage, and your job is to select which ones you think are "Wudsy." Afterward, the aliens tell you which shapes are "Wudsy."

The meaning of the word Wudsy is the same during the whole experiment. However, it is possible that whether something is Wudsy depends on what other shapes it is in the context of. Wudsy may or may not correspond to an English word.

To start with, no one has given you any examples of what counts as "Wudsy." So just do your best below and pick which ones you think might belong to the concept called "Wudsy." Right after you do so, the aliens are going to label the Wudsy objects by drawing a black box around them, and then you are going to get another round of guessing which objects are "Wudsy."

You will go through 15 trials of guessing what counts as Wudsy. Remember that the meaning of Wudsy does not change during the experiment, but it might depend on the other shapes in the collection.

(trial 1/15) Click yes on the objects that you think are Wudsy, and No on the other objects. Then click Next.

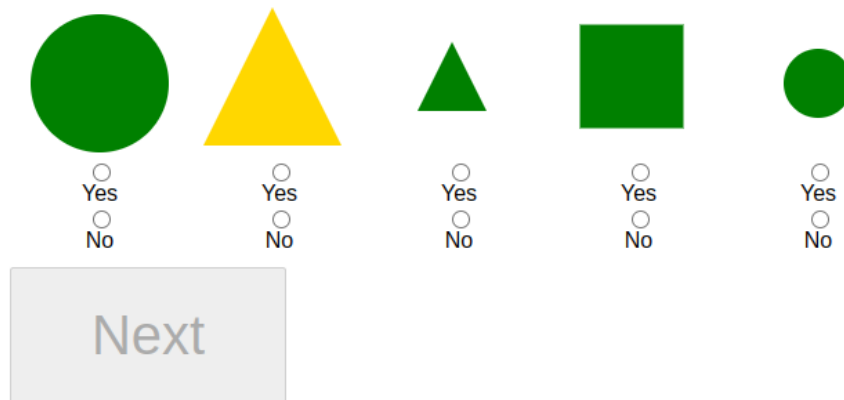


Figure 8: Cover story and web interface for our version of the logical concept learning study, which is based on [45]

486 A.3.2 Parameter fitting

487 Training consists of fitting the parameters T , θ (for the prior), ϵ (for the likelihood), α and β (for the
488 logical concepts likelihood), and Platt transform parameters a , b (for the Number Game). In practice,
489 this amounts to around 400 parameters, almost all of which come from θ .

490 We fit these parameters using Adam with a learning rate of 0.001. We perform 1000 epochs of
491 training for the Number Game, and 100 epochs for logical concepts. There is a tenfold difference in
492 the number of concepts, so this way they take about the same number of gradient steps.

493 For the number game we do 10-fold cross validation to calculate holdout predictions. For logical
494 concepts we use the train-test split introduced in [45], which involves running different groups of
495 human subjects on each concept twice, with different random examples. One sequence of random
496 examples is arbitrarily designated as training data, and the other as holdout data.

497 All model were trained on a laptop using no GPUs. Training takes between a few minutes and an
498 hour, depending on the domain and the model.

499 Some of the parameters that we fit, namely ϵ , α , β , cannot be negative. To enforce this we actually
500 optimize the inverse logistic of those parameters.

501 A.3.3 MCMC over Logical Expressions

502 Fleet was used¹ to perform MCMC over logical expressions with the domain-specific primitives in
503 this file, which include:

```

    true, false : boolean
    blue, yellow, green : object → boolean
    rectangle, circle, triangle : object → boolean
    small, medium, large : object → boolean
    and, or,  $\iff$ ,  $\implies$  : boolean × boolean → boolean
     $\forall$ ,  $\exists$  : (shape → boolean) ×  $2^{\text{object}}$  → boolean
    filter : (object → boolean) ×  $2^{\text{object}}$  →  $2^{\text{object}}$ 
     $\in$  : object ×  $2^{\text{object}}$  → boolean
     $\iota$  :  $2^{\text{object}}$  → object ∪ {⊥}, unique set element
    empty :  $2^{\text{object}}$  → boolean
    same_shape, same_color, same_size : object × object → boolean
    size<, size≤, size>, size≥, : object × object → boolean
```

504 The model first constructed a large hypothesis space by performing MCMC for 1 minute per batch,
505 and per learning curve. In one minute, Fleet makes approximately 10^6 MH proposals. There are a
506 little more than 200 learning curves, and 25 batches per curve, for a total of about 5 billion MCMC
507 proposals. In the main text, we abbreviate this analysis by referring to 10^9 proposals.

508 The top 10 samples per batch and per learning curve were retained. These top 10 samples samples
509 were then deduplicated to yield 45 thousand hypotheses. Parameter fitting and posterior estimation
510 was performed solely over those 45 thousand hypotheses.

511 Quantitatively, these are vastly more proposals than the models introduced in this paper. Quantitatively,
512 these proposals are also derived in a very different way: the hypothesis space for the BPL learner is
513 actually informed by data on other learning curves, and also on the same learning curve, but in the
514 future batches.

515 It is in this sense that the BPL model is a computational-level theory, and not a process model,
516 because human subjects could not be proposing hypotheses using data that is going to be seen in the
517 future, or on other learning curves. However, the above strategy for proposing hypotheses is a very
518 reasonable heuristic for constructing the support of the space of plausible logical hypotheses that a
519 human learner might ever think of.

520 A.4 Prompting

521 A.4.1 Proposing hypotheses

522 For the number game we use the following prompt for code-davinci-002 to generate candidate
523 concepts in natural language, given examples $X_{1:K}$. The example number concepts given in the
524 prompt come from the cover score given to human participants [33]:

```
525 # Python 3
526 # Here are a few example number concepts:
527 # -- The number is even
528 # -- The number is between 30 and 45
529 # -- The number is a power of 3
530 # -- The number is less than 10
531 #
532 # Here are some random examples of numbers belonging to a different ↗
533 ↘ number concept:
534 #  $X_{1:K}$ 
```

¹Running the model was graciously performed by the authors of [45], who provided us with the raw data.

```

535 # The above are examples of the following number concept:
536 # -- The number is

537 where  $X_{1:K}$  is formatted by listing the numbers with a comma and a space between them.

538 For the number game we used the following prompt to generate candidate concepts in python (code
539 baseline):

540 # Python 3
541 # Here are a few example number concepts:
542 # -- The number is even
543 # -- The number is between 30 and 45
544 # -- The number is a power of 3
545 # -- The number is less than 10
546 #
547 # Here are some random examples of numbers belonging to a different ↗
548 ↘ number concept:
549 #  $X_{1:K}$ 
550 # Write a python function that returns true if 'num' belongs to ↗
551 ↘ this number concept.
552 def check_if_in_concept(num):
553     return

554 For logical concepts we used the following few-shot prompt for GPT-4 to generate candidate concepts:

555 Here three simple concepts, which specify when an object is ↗
556 ↘ 'positive' relative to an example collection of other ↗
557 ↘ objects. Before giving the rule for each concept, we give ↗
558 ↘ examples of collections of objects, and which objects in the ↗
559 ↘ collection are 'positive'.
560
561 Concept #1:
562     An Example of Concept #1:
563         POSITIVES: (big yellow rectangle)
564         NEGATIVES: (big green circle), (medium yellow rectangle)
565     Another Example of Concept #1:
566         POSITIVES: (medium yellow rectangle)
567         NEGATIVES: (big red circle), (small green circle)
568 Rule for Concept #1: Something is positive if it is the biggest ↗
569 ↘ yellow object in the example.
570
571
572 Concept #2:
573     An Example of Concept #2:
574         POSITIVES: (small yellow circle), (medium yellow rectangle)
575         NEGATIVES: (big green circle), (big blue rectangle)
576     Another Example of Concept #2:
577         POSITIVES: (big blue circle), (medium blue rectangle)
578         NEGATIVES: (small green circle), (medium yellow rectangle),
579 Rule for Concept #2: Something is positive if there is another ↗
580 ↘ object with the same color in the example.
581
582 Concept #3:
583     An Example of Concept #3:
584         POSITIVES: (small yellow circle), (medium yellow rectangle)
585         NEGATIVES: (big green circle), (big blue rectangle)
586     Another Example of Concept #3:
587         POSITIVES: (small blue circle), (small blue triangle), ↗
588         ↘ (medium blue rectangle)
589         NEGATIVES: (medium green triangle), (big yellow rectangle)
590     Another Example of Concept #3:
591         POSITIVES: (big red rectangle), (medium red rectangle), ↗
592         ↘ (big red triangle)
593         NEGATIVES: (medium green triangle), (big yellow rectangle)
594 Rule for Concept #3: Something is positive if it is the same color ↗
595 ↘ as the smallest triangle in the example.

```

596
597 Now here are some examples of another concept called Concept #4, ↵
598 ↵ but this time we don't know the rule. Infer ten different ↵
599 ↵ possible rules, and make those ten rules as simple and ↵
600 ↵ general as you can. Your simple general rules might talk ↵
601 ↵ about shapes, colors, and sizes, and might make comparisons ↵
602 ↵ between these features within a single example, but it ↵
603 ↵ doesn't have to. Remember that a rule should say when ↵
604 ↵ something is positive, and should mention the other objects ↵
605 ↵ in the example, and should be consisting with what you see ↵
606 ↵ below.

607
608 Concept #4:
609 $X_{1:K}$

610 Rule for Concept #4: Something is positive if...

611
612 Now make a numbered list of 10 possible rules for Concept #4. Start ↵
613 ↵ by writing "1. Something is positive if". End each line with ↵
614 ↵ a period.

615 Each sample from the above prompt generates 10 possible concepts formatted as a numbered list. We
616 draw 10 times at temperature=1 to construct 100 hypotheses. To obtain fewer than 100 hypotheses we
617 take hypotheses from each sampled list in round-robin fashion. We found that asking it to generate
618 a list of hypotheses generated greater diversity without sacrificing quality, compared to repeatedly
619 sampling a single hypothesis.

620 The above prompt provides in-context examples of first-order rules. We also tried using a different
621 prompt for propositional concepts that illustrated the examples as a truth table, and gave in-context
622 example rules that were propositional:

623 Here are some example concepts defined by a logical rule:
624
625 Rule: a triangle.
626 Rule: a green rectangle.
627 Rule: big or a rectangle (unless that rectangle is blue).
628 Rule: not both big and green.
629 Rule: either big or green, but not both.
630 Rule: either a rectangle or not yellow.
631 Rule: a circle.
632
633

634 Now please produce a logical rule for a new concept. Your rule ↵
635 ↵ should be consistent with the following examples. It must be ↵
636 ↵ true of all the positive examples, and not true of all the ↵
637 ↵ negative examples. The examples are organized into a table ↵
638 ↵ with one column for each feature (size, color, shape):

639
640 $X_{1:K}$
641

642 Please produce a simple rule that is consistent with the above ↵
643 ↵ table. Make your rule as SHORT, SIMPLE, and GENERAL as ↵
644 ↵ possible. Do NOT make it more complicated than it has to be, ↵
645 ↵ or refer to features that you absolutely do not have to refer ↵
646 ↵ to. Begin by writing "Rule: " and then the rule, followed by ↵
647 ↵ a period.

648 Using the first order prompt for every concept gives a $R^2 = .80$ fit to the human responses. Using both
649 prompts gives the $R^2 = .81$ result in the main paper: the propositional prompt for the propositional
650 problems, and the first order prompt for the higher order problems. We strongly suspect that a single
651 prompt that just showed both propositional and higher-order in-context examples would work equally
652 well, given that a single first-order prompt works about as well also, but we did not try that because
653 of the high cost of using GPT-4.

654 On the first batch, the learner has not observed any examples. Therefore the above prompts do not
655 apply, and we use a different prompt to construct an initial hypothesis space:

656 Here are some example concepts defined by a logical rule:

657

658 Rule: color is purple.

659 Rule: shape is not a hexagon.

660 Rule: color is purple and size is small.

661 Rule: size is tiny or shape is square.

662 Rule: size is not enormous.

663 Rule: color is red.

664

665 Please propose a some new concepts, defined by a logical rule. ↵

666 ↳ These new concepts can only refer to the following features:

667 - shape: triangle, rectangle, circle

668 - color: green, blue, yellow

669 - size: small, medium, large

670

671 Please make your rules short and simple, and please write your ↵

672 ↳ response on a single line that begins with the text "Rule: ". ↵

673 ↳ Provide 100 possible rules.

674 We generate from the above prompt at temperature=0, and split on line breaks to obtain candidate
675 rules.

676 A.4.2 Translating from natural language to Python

677 We translate Number Game concepts from English to Python via the following prompt for code-
678 davinci-002, and generate at temperature=0 until linebreak:

```
679 # Write a python function to check if a number is C.
```

```
680 def check_number(num):
```

```
681     return
```

682 We translate the logic cool concepts from English to Python using a series of in-context examples,
683 again generating with temperature=0 until the text #DONE is produced.²

```
684 def check_object(this_object, other_objects):
```

```
685     """
```

```
686         this_object: a tuple of (shape, color, size)
```

```
687         other_objects: a list of tuples of (shape, color, size)
```

```
688
```

```
689         returns: True if 'this_object' is positive according to the ↵
```

```
690             ↳ following rule:
```

```
691                 Something is positive if it is not a small object, and not ↵
```

```
692                 ↳ a green object.
```

```
693     """
```

```
694     # shape: a string, either "circle", "rectangle", or "triangle"
```

```
695     # color: a string, either "yellow", "green", or "blue"
```

```
696     # size: an int, either 1 (small), 2 (medium), or 3 (large)
```

```
697     this_shape, this_color, this_size = this_object
```

```
698
```

```
699     # 'this_object' is not a part of 'other_objects'
```

```
700     # to get all of the examples, you can use ↵
```

```
701         ↳ 'all_example_objects', defined as 'other_objects + ↵
```

```
702         ↳ [this_object]'
```

```
703     # be careful as to whether you should be using ↵
```

```
704         ↳ 'all_example_objects' or 'other_objects' in your code
```

```
705     all_example_objects = other_objects + [this_object]
```

```
706
```

```
707     # Something is positive if it is not a small object, and not a ↵
```

```
708     ↳ green object.
```

²This prompt is pretty long, and probably could be much shorter. Preliminary experiments suggested that a few in-context examples were very helpful, and so to increase the odds of the model working without much time spent prompt-engineering, we provided a large number of in-context examples.

```

709     #START
710     return (not this_size == 1) and (not this_color == "green")
711 #DONE
712
713 def check_object(this_object, other_objects):
714     """
715     this_object: a tuple of (shape, color, size)
716     other_objects: a list of tuples of (shape, color, size)
717
718     returns: True if 'this_object' is positive according to the ↵
719             ↳ following rule:
720             ↳ Something is positive if it is bigger than every other object
721     """
722     # shape: a string, either "circle", "rectangle", or "triangle"
723     # color: a string, either "yellow", "green", or "blue"
724     # size: an int, either 1 (small), 2 (medium), or 3 (large)
725     this_shape, this_color, this_size = this_object
726
727     # 'this_object' is not a part of 'other_objects'
728     # to get all of the examples, you can use ↵
729     ↳ 'all_example_objects', defined as 'other_objects + ↵
730     ↳ [this_object]'
731     # be careful as to whether you should be using ↵
732     ↳ 'all_example_objects' or 'other_objects' in your code
733     all_example_objects = other_objects + [this_object]
734
735     # Something is positive if it is bigger than every other object
736     #START
737     return all( this_size > other_object[2] for other_object in ↵
738               ↳ other_objects )
739 #DONE
740
741 def check_object(this_object, other_objects):
742     """
743     this_object: a tuple of (shape, color, size)
744     other_objects: a list of tuples of (shape, color, size)
745
746     returns: True if 'this_object' is positive according to the ↵
747             ↳ following rule:
748             ↳ Something is positive if it is one of the largest
749     """
750     # shape: a string, either "circle", "rectangle", or "triangle"
751     # color: a string, either "yellow", "green", or "blue"
752     # size: an int, either 1 (small), 2 (medium), or 3 (large)
753     this_shape, this_color, this_size = this_object
754
755     # 'this_object' is not a part of 'other_objects'
756     # to get all of the examples, you can use ↵
757     ↳ 'all_example_objects', defined as 'other_objects + ↵
758     ↳ [this_object]'
759     # be careful as to whether you should be using ↵
760     ↳ 'all_example_objects' or 'other_objects' in your code
761     all_example_objects = other_objects + [this_object]
762
763     # Something is positive if it is one of the largest
764     #START
765     return all( this_size >= other_object[2] for all_example_object ↵
766               ↳ in all_example_objects )
767 #DONE
768
769
770 def check_object(this_object, other_objects):
771     """
772     this_object: a tuple of (shape, color, size)
773     other_objects: a list of tuples of (shape, color, size)

```



```

774
775     returns: True if 'this_object' is positive according to the ↵
776         ↳ following rule:
777         ↳ Something is positive if it is smaller than every yellow ↵
778             ↳ object
779     """
780     # shape: a string, either "circle", "rectangle", or "triangle"
781     # color: a string, either "yellow", "green", or "blue"
782     # size: an int, either 1 (small), 2 (medium), or 3 (large)
783     this_shape, this_color, this_size = this_object
784
785     # 'this_object' is not a part of 'other_objects'
786     # to get all of the examples, you can use ↵
787         ↳ 'all_example_objects', defined as 'other_objects + ↵
788             ↳ [this_object]'
789     # be careful as to whether you should be using ↵
790         ↳ 'all_example_objects' or 'other_objects' in your code
791     all_example_objects = other_objects + [this_object]
792
793     # Something is positive if it is smaller than every yellow object
794     #START
795     return all( this_size < other_object[2] for other_object in ↵
796         ↳ other_objects if other_object[1] == "yellow" )
797 #DONE
798
799 def check_object(this_object, other_objects):
800     """
801     this_object: a tuple of (shape, color, size)
802     other_objects: a list of tuples of (shape, color, size)
803
804     returns: True if 'this_object' is positive according to the ↵
805         ↳ following rule:
806         ↳ Something is positive if there is another object with the ↵
807             ↳ same color
808     """
809     # shape: a string, either "circle", "rectangle", or "triangle"
810     # color: a string, either "yellow", "green", or "blue"
811     # size: an int, either 1 (small), 2 (medium), or 3 (large)
812     this_shape, this_color, this_size = this_object
813
814     # 'this_object' is not a part of 'other_objects'
815     # to get all of the examples, you can use ↵
816         ↳ 'all_example_objects', defined as 'other_objects + ↵
817             ↳ [this_object]'
818     # be careful as to whether you should be using ↵
819         ↳ 'all_example_objects' or 'other_objects' in your code
820     all_example_objects = other_objects + [this_object]
821
822     # Something is positive if there is another object with the ↵
823         ↳ same color
824     #START
825     return any( this_color == other_object[1] for other_object in ↵
826         ↳ other_objects )
827 #DONE
828
829 def check_object(this_object, other_objects):
830     """
831     this_object: a tuple of (shape, color, size)
832     other_objects: a list of tuples of (shape, color, size)
833
834     returns: True if 'this_object' is positive according to the ↵
835         ↳ following rule:
836         ↳ Something is positive if it has a unique combination of ↵
837             ↳ color and shape
838     """

```

```

839 # shape: a string, either "circle", "rectangle", or "triangle"
840 # color: a string, either "yellow", "green", or "blue"
841 # size: an int, either 1 (small), 2 (medium), or 3 (large)
842 this_shape, this_color, this_size = this_object
843
844 # 'this_object' is not a part of 'other_objects'
845 # to get all of the examples, you can use ↵
846     ↵ 'all_example_objects', defined as 'other_objects + ↵
847     ↵ [this_object]'
848 # be careful as to whether you should be using ↵
849     ↵ 'all_example_objects' or 'other_objects' in your code
850 all_example_objects = other_objects + [this_object]
851
852 # Something is positive if it has a unique combination of color ↵
853     ↵ and shape
854 #START
855 return all( this_shape != other_object[0] or this_color != ↵
856     ↵ other_object[1] for other_object in other_objects )
857 #DONE
858
859 def check_object(this_object, other_objects):
860     """
861     this_object: a tuple of (shape, color, size)
862     other_objects: a list of tuples of (shape, color, size)
863
864     returns: True if 'this_object' is positive according to the ↵
865         ↵ following rule:
866         ↵ Something is positive if it has the same color as the ↵
867         ↵ majority of objects
868     """
869     # shape: a string, either "circle", "rectangle", or "triangle"
870     # color: a string, either "yellow", "green", or "blue"
871     # size: an int, either 1 (small), 2 (medium), or 3 (large)
872     this_shape, this_color, this_size = this_object
873
874     # 'this_object' is not a part of 'other_objects'
875     # to get all of the examples, you can use ↵
876         ↵ 'all_example_objects', defined as 'other_objects + ↵
877         ↵ [this_object]'
878     # be careful as to whether you should be using ↵
879         ↵ 'all_example_objects' or 'other_objects' in your code
880     all_example_objects = other_objects + [this_object]
881
882     # Something is positive if it has the same color as the ↵
883         ↵ majority of objects
884     #START
885     majority_color = max(["yellow", "green", "blue"], key=lambda ↵
886         ↵ color: sum(1 for obj in all_example_objects if obj[1] == ↵
887         ↵ color))
888     return this_color == majority_color
889 #DONE
890
891 def check_object(this_object, other_objects):
892     """
893     this_object: a tuple of (shape, color, size)
894     other_objects: a list of tuples of (shape, color, size)
895
896     returns: True if 'this_object' is positive according to the ↵
897         ↵ following rule:
898         ↵ Something is positive if there are at least two other ↵
899         ↵ objects with the same shape
900     """
901     # shape: a string, either "circle", "rectangle", or "triangle"
902     # color: a string, either "yellow", "green", or "blue"
903     # size: an int, either 1 (small), 2 (medium), or 3 (large)

```

```

904     this_shape, this_color, this_size = this_object
905
906     # 'this_object' is not a part of 'other_objects'
907     # to get all of the examples, you can use ↵
908         ↵ 'all_example_objects', defined as 'other_objects + ↵
909         ↵ [this_object]'
910     # be careful as to whether you should be using ↵
911         ↵ 'all_example_objects' or 'other_objects' in your code
912     all_example_objects = other_objects + [this_object]
913
914     # Something is positive if there are at least two other objects ↵
915         ↵ with the same shape
916     #START
917     return sum(1 for other_object in other_objects if ↵
918         ↵ other_object[0] == this_shape) >= 2
919 #DONE
920
921 def check_object(this_object, other_objects):
922     """
923     this_object: a tuple of (shape, color, size)
924     other_objects: a list of tuples of (shape, color, size)
925
926     returns: True if 'this_object' is positive according to the ↵
927         ↵ following rule:
928         C
929     """
930     # shape: a string, either "circle", "rectangle", or "triangle"
931     # color: a string, either "yellow", "green", or "blue"
932     # size: an int, either 1 (small), 2 (medium), or 3 (large)
933     this_shape, this_color, this_size = this_object
934
935     # 'this_object' is not a part of 'other_objects'
936     # to get all of the examples, you can use ↵
937         ↵ 'all_example_objects', defined as 'other_objects + ↵
938         ↵ [this_object]'
939     # be careful as to whether you should be using ↵
940         ↵ 'all_example_objects' or 'other_objects' in your code
941     all_example_objects = other_objects + [this_object]
942
943     # C
944     #START

```

945 A.5 GPT-4 Baselines

946 Our GPT-4 baseline for each domain presented the examples $X_{1:K}$ in string form and then asked
947 GPT-4 to respond Yes/No as to whether a test example X_{test} belonged to the same concept. GPT-4
948 was then queried at temperature=1 to collect 10 samples. Samples not beginning with 'y'/'n' were
949 discarded, and the ratio of remaining samples that began with 'y' was computed (case insensitive).

950 We show below example prompts for the number and logic domains.

951 Here are a few example number concepts:

```

952 -- The number is even
953 -- The number is between 30 and 45
954 -- The number is a power of 3
955 -- The number is less than 10
956

```

957 Here are some random examples of numbers belonging to a possibly ↵
958 ↵ different number concept:

959 98, 81, 86, 93

960

961 Question: Does the number 42 belong to the same concept as the ↵
962 ↵ above numbers?

963 Answer (one word, yes/no):

964 Logical concept example prompt:

965 Here are some example concepts defined by a logical rule:

966

967 Rule for Concept #1: Something is positive if it is the biggest ↗
968 ↳ yellow object in the example

969 Rule for Concept #2: Something is positive if there is another ↗
970 ↳ object with the same color in the example

971 Rule for Concept #3: Something is positive if it is the same color ↗
972 ↳ as the smallest triangle in the example

973

974 Now please look at the following examples for a new logical rule.

975

976 An Example of Concept #4:
977 POSITIVES: none
978 NEGATIVES: (large yellow circle), (small green circle), ↗
979 ↳ (medium green circle), (small yellow triangle)

980 Another Example of Concept #4:
981 POSITIVES: (small green circle), (large green circle)
982 NEGATIVES: (large yellow circle), (medium blue circle)

983 Another Example of Concept #4:
984 POSITIVES: (small green rectangle)
985 NEGATIVES: (medium yellow circle), (medium blue rectangle), ↗
986 ↳ (large green circle), (medium green circle)

987 Another Example of Concept #4:
988 POSITIVES: (medium green rectangle)
989 NEGATIVES: (medium yellow circle), (small yellow ↗
990 ↳ rectangle), (medium yellow rectangle), (medium blue ↗
991 ↳ rectangle)

992 Another Example of Concept #4:
993 POSITIVES: (small green rectangle)
994 NEGATIVES: (large yellow rectangle), (small yellow ↗
995 ↳ triangle), (medium green circle), (small blue rectangle)

996 Another Example of Concept #4:
997 POSITIVES: (medium green triangle)
998 NEGATIVES: (medium blue triangle), (medium blue rectangle), ↗
999 ↳ (large blue triangle), (small yellow triangle)

1000 Another Example of Concept #4:
1001 POSITIVES: none
1002 NEGATIVES: (small yellow circle), (large blue circle)

1003 Another Example of Concept #4:
1004 POSITIVES: none
1005 NEGATIVES: (large green circle), (small blue rectangle), ↗
1006 ↳ (small green triangle), (medium blue rectangle)

1007 Another Example of Concept #4:
1008 POSITIVES: (small green rectangle)
1009 NEGATIVES: (small yellow circle), (large blue rectangle)

1010

1011 Now we get a new collection of examples for Concept #4:
1012 (medium blue triangle) (large yellow triangle) (small blue ↗
1013 ↳ rectangle) (large blue circle) (small yellow circle)

1014 Question: Based on the above example, is a (small yellow circle) in ↗
1015 ↳ the concept?

1016 Answer (one word, just write yes/no):

1017 A.6 Latent Language Baseline

1018 For fair comparison, we designed our latent language baseline to be as similar to our system as
1019 possible. It performs maximum likelihood estimation of a single concept, rather than estimate a
1020 full posterior, but uses the exact same prompts and likelihood functions as our model. The most
1021 important difference from the original latent language paper [22] is that instead of training our
1022 own neural models for language interpretation and language generation, we use pretrained models
1023 (Codex/code-davinci-002 and GPT-4).

1024 A.7 Ablation of the proposal distribution

1025 We ablate the proposal distribution by proposing hypotheses unconditioned on $X_{1:K}$. We accomplish
1026 this by drawing concepts from the following alternative prompt, which is designed to resemble the
1027 prompt used by the full model except that it does not include $X_{1:K}$:

```
1028 # Python 3
1029 # Here are a few example number concepts:
1030 # -- The number is even
1031 # -- The number is between 30 and 45
1032 # -- The number is a power of 3
1033 # -- The number is less than 10
1034 # -- The number is
```

1035 A.8 Pretrained prior

1036 Our pretrained prior comes from the opensource model CodeGen [34], which was trained on source
1037 code. We chose this model because we suspected that pretraining on source code would give better
1038 density estimation for text describing precise rules. We formatted the rules as a natural language
1039 comment and prefixed it with a small amount of domain-specific text in order to prime the model to
1040 put probability mass on rules that correctly talk about numbers or shapes.

1041 For the number game, we would query CodeGen for the probability of $p(C)$ via

```
1042 # Here is an example number concept:
1043 # The number is  $C$ 
```

1044 For the number game’s code baseline, we would query CodeGen for the probability of $p(C)$ via

```
1045 # Python 3
1046 # Let’s think of a number concept.
1047 # Write a python function that returns true if ‘num’ belongs to ↯
1048     ↯ this number concept.
1049 def check_if_in_concept(num):
1050     return  $C$ 
```

1051 For logical concepts we would query CodeGen for the probability of $p(C)$ via

```
1052 # Here are some simple example shape concepts:
1053 # 1. neither a triangle nor a green rectangle
1054 # 2. not blue and large.
1055 # 3. if it is large, then it must be yellow.
1056 # 4. small and blue
1057 # 5. either big or green.
1058 # 6.  $C$ 
```

1059 Because the proposal distribution would generate rules beginning with the prefix “Something is
1060 positive if...” we would remove that text before computing $p(C)$ as above.