December 20, 2024

0.1 Memory Task on LLM - Nimish Jain

0.1.1 Task Description

Notes from Paper: https://psycnet.apa.org/manuscript/2018-17847-005.pdf

The results from a study in which 453 participants took part in five different memory tasks: single-item recognition, associative recognition, cued recall, free recall, and lexical decision.

Among other things, we find that: * (a) the processes involved in lexical access and episodic memory are largely separate and rely on different kinds of information; * (b) access to lexical memory is driven primarily by perceptual aspects of a word; * (c) all episodic memory tasks rely to an extent on a set of shared processes which make use of semantic features to encode both single words and associations between words; * (d) recall involves additional processes likely related to contextual cuing and response production.

These results provide a large-scale picture of memory across different tasks, which can serve to drive the development of comprehensive theories of memory.

The purpose of the present work is to examine how performance on different memory tasks is correlated with respect to both the processes engaged by individual participants and the information conveyed by particular item characteristics (intelligence, engagement, motivation, etc.). This is why it is critical to examine not just the pairwise correlations among tasks but the entire pattern of correlations among many tasks in order to better identify meaningful correlations and reject spurious ones. The present study is the first to jointly examine the patterns of **memory correlations across tasks, items, and individuals**.

For example: * Each participant would complete several blocks of free recall, some with lists of random words, some with lists of paired semantic associates, and some with lists consisting of exemplars of a single category (paired associates, free recall, memory span, recognition, and verbal discrimination).

The experiment included five different tasks: single-item recognition, associative recognition, cued recall, free recall, and lexical decision. Each task was repeated three times over the course of the experiment for a total of 15 blocks, with 20 test trials per block. The first five blocks consisted of the first presentation of each of the five tasks (randomly ordered for each participant). For the remaining 10 blocks, the five task types were presented twice in random order. The task was post-cued; therefore, participants could not adopt a study strategy based on the anticipated test type.

The items in each block were randomly sampled from the pool of 924 words without replacement for each participant, such that no items repeated between blocks for a given participant. All blocks (except lexical decision) began with a study phase where participants viewed 20 word pairs presented side by side, one pair at a time. Each pair remained on the screen for 2 seconds and was immediately followed by asking participants to "Please rate the degree of association between the two items you just saw" on a scale from 1–9 where 1 is "not at all associated" and 9 is "highly associated." The word pair was not visible on the screen during the rating. Responses were self-paced by clicking on boxes numbered 1–9 on the screen.

Each study phase was immediately followed by a distractor task. This was a simple math task where participants continuously added a series of 15 random digits drawn with replacement from the range 1–9. Digits were presented at a rate of 3 seconds per digit, for a total presentation time of 45 seconds. After all digits appeared, participants typed in their response and received accuracy feedback.

Following the distractor task, participants were presented with one of the following memory tasks. Responses in all tasks were self-paced. Each study/test block was followed by the option to take a self-paced break. The experiment lasted approximately one hour.

Single-item recognition: For the target stimuli, ten study items were selected at random from the study list. The ten items could be from either the right or the left presentation position, but not from both the left and the right presentation position for the same study trial. In other words, only one of the words in the study word pair could be selected. These ten old items were combined with 10 foils and presented in random order in the center of the screen. Participants were asked to "indicate if the item you see on the screen was on the list you just studied (YES) or not on the list (NO)". Participants responded by clicking on boxes presented on the computer screen.

Associative recognition: For the test stimuli, ten word pairs were selected at random from the study list. The remaining ten word pairs were scrambled such that none of the pairs remained intact. The scrambled pairs could be rearranged both between earlier and later study positions as well as between right and left presentation positions. The ten intact word pairs were combined with the ten rearranged word pairs and presented in random order in the center of the screen with one word appearing above the other rather than side by side. Participants were asked to "indicate if the PAIR of words you see on the screen was studied as a PAIR on the list you just studied (YES) or not a pair (NO)". Participants responded by clicking on boxes presented on the computer screen.

Cued recall: For the test stimuli, twenty study items were selected from the study list, one from each pair. Ten of the twenty items were from the right study presentation position and ten were from the left study presentation position, randomly chosen. In this way, all twenty study pairs are tested, but half the cue words were from the right presentation position and half were from the left presentation position. The twenty cue words were presented in random order to the left of a box on the computer screen where participants were asked to enter the corresponding word in the pair. Participants were asked to respond by "typing the OTHER WORD in the pair. For example, if you studied BRICK BRACK and you now see BRICK your response should be BRACK. If you cannot recall the word, click DON'T REMEMBER". It was emphasized that spelling did not matter; rather they should focus on providing as many responses as possible.

Free recall: No words were presented at test, rather participants were asked to "try to recall as many words from the study list as you possibly can. When you cannot recall any more words, click on the FINISHED button". Participants were required to attempt to provide responses for a minimum of 90 seconds. A timer appeared on the screen, and the finished button could not be clicked until 90 seconds had passed. It was emphasized that spelling did not matter; rather they should focus on providing as many responses as possible.

Lexical decision: This task was not preceded by a study block. For the test stimuli, ten words drawn from the complete word set were combined with 10 pseudo-words and presented in random order in the center of the screen. Participants were simply presented with a word and asked to "indicate if the item you see is a word (YES) or not a word (NO). Respond as QUICKLY as possible". Participants responded "word" by clicking the left mouse button and "non-word" by clicking the right mouse button. Response time was measured from the onset of the word to the click of the mouse button.

0.1.2 Importing Packages

[1]:
```
import pandas as pd
```

0.1.3 Dataset Loading

[2]:
```
# Step 1: Load and inspect the dataset
file_path = '/content/all_data.csv'
data = pd.read_csv(file_path)
data.head()
```

```
<ipython-input-2-224824ce77cc>:3: DtypeWarning: Columns (8,9,31,32,36) have
mixed types. Specify dtype option on import or set low_memory=False.
  data = pd.read_csv(file_path)
```

[2]:
```
   Unnamed: 0  subject  block phase                 condition  trial  \
0        37.0        2      1  test  Associative recognition      1
1        38.0        2      1  test  Associative recognition      2
2        39.0        2      1  test  Associative recognition      3
3        40.0        2      1  test  Associative recognition      4
4        41.0        2      1  test  Associative recognition      5

   stim.num.left  stim.num.right stim.string.left stim.string.right  …  \
0            141             722            CLAIM         SCATTERED  …
1            809             748          SHARPLY            STRESS  …
2            894             352           FORMER            VOLUME  …
3            683             189         CONTROLS         REMAINING  …
4            857              30            TRACK        APPARENTLY  …

   study.partner.left.num  study.partner.right.num  study.partner.left.string  \
0                   722.0                    141.0                  SCATTERED
1                   747.0                    620.0                     SHARED
2                   352.0                    894.0                     FORMER
3                   919.0                    737.0                    WRAPPED
4                   665.0                    122.0                     READER

   study.partner.right.string  resp.num  study.partner.resp.num  \
0                       CLAIM       NaN                     NaN
```

```
1                    POURED       NaN                      NaN
2                    VOLUME       NaN                      NaN
3                 SENTENCES       NaN                      NaN
4                      CASH       NaN                      NaN

   study.partner.resp.string  recall.type  recall.list  \
0                        NaN          NaN          NaN
1                        NaN          NaN          NaN
2                        NaN          NaN          NaN
3                        NaN          NaN          NaN
4                        NaN          NaN          NaN

            condition.block
0  Associative recognition 1
1  Associative recognition 1
2  Associative recognition 1
3  Associative recognition 1
4  Associative recognition 1

[5 rows x 39 columns]
```

[3]:
```
# Inspect the dataset structure and columns
print("Dataset Columns:", data.columns)
```

```
Dataset Columns: Index(['Unnamed: 0', 'subject', 'block', 'phase', 'condition',
'trial',
       'stim.num.left', 'stim.num.right', 'stim.string.left',
       'stim.string.right', 'stim.distractor', 'studied', 'freq.left',
       'cv.left', 'old.left', 'freq.right', 'cv.right', 'old.right', 'rt',
       'distractor.resp', 'resp', 'resp.string', 'resp.type',
       'resp.type.rescore', 'study.pos.left', 'study.pos.right', 'kf.left',
       'kf.right', 'resp.string.corr', 'study.partner.left.num',
       'study.partner.right.num', 'study.partner.left.string',
       'study.partner.right.string', 'resp.num', 'study.partner.resp.num',
       'study.partner.resp.string', 'recall.type', 'recall.list',
       'condition.block'],
      dtype='object')
```

[4]:
```
print("Number of Rows in the data:", data.shape[0])
```

```
Number of Rows in the data: 168107
```

[5]:
```
print("Condition Blocks types: ", data['condition'].unique())
```

```
Condition Blocks types:  ['Associative recognition' 'Free recall' 'Cued recall'
'Lexical decision'
 'Single recognition']
```

Data Columns Explanation

Unnamed: 0: An index column, likely a row number from the original dataset.

subject: The unique identifier for a participant in the experiment.

block: Represents a specific block of trials in the experiment (there are 15 blocks per participant as described).

phase: Indicates the phase of the experiment (e.g., study, test, distractor, etc.).

condition: The experimental condition under which the task was performed.

trial: The specific trial within a block.

stim.num.left / stim.num.right: Numerical identifiers for the stimuli presented on the left/right in a pair.

stim.string.left / stim.string.right: The actual word strings presented on the left/right in a pair.

stim.distractor: Indicates if the stimulus is a distractor.

studied: Indicates whether the stimulus was part of the study phase.

freq.left / freq.right: Word frequency measures for the left/right words (how commonly the word is used in the language).

cv.left / cv.right: Likely measures of concreteness or variability for the left/right words.

old.left / old.right: Indicates if the left/right word is an "old" item (presented before in the study phase).

rt: Reaction time, the time taken by the participant to respond in the trial.

distractor.resp: Participant's response during the distractor task.

resp: Participant's raw response (numerical or binary).

resp.string: The string version of the participant's response.

resp.type: Type of response (e.g., correct, incorrect).

resp.type.rescore: Rescored response type (manually corrected or validated for errors).

study.pos.left / study.pos.right: Study position of the left/right words during the study phase.

kf.left / kf.right: Likely other characteristics of the left/right words (e.g., concreteness ratings, frequency measures).

study.partner.left.num / study.partner.right.num: Numerical identifier for the partner word in the pair for left/right words.

study.partner.left.string / study.partner.right.string: String representation of the partner word for left/right words.

resp.string.corr: Corrected response string (after handling typos or close matches).

study.partner.resp.num / study.partner.resp.string: Numerical or string version of the response word paired with the study word.

recall.type: Type of recall task (e.g., free recall, cued recall).

recall.list: List identifier for the words to recall in that block or trial.

condition.block: A combination of condition and block, representing a unique experimental scenario.

0.1.4 Human Accuracy Calculation

0.1.5 1. Single Recognition Condition

Extract relevant data

[6]:
```python
single_recognition_condition = data[data['condition'] == 'Single recognition']
single_recognition_condition.head()
```

[6]:
```
    Unnamed: 0  subject  block phase           condition  trial  \
73       218.0        2      5  test  Single recognition      1
74       219.0        2      5  test  Single recognition      2
75       220.0        2      5  test  Single recognition      3
76       221.0        2      5  test  Single recognition      4
77       222.0        2      5  test  Single recognition      5

    stim.num.left  stim.num.right stim.string.left stim.string.right  …  \
73            643               0            PROVE               NaN  …
74            497               0            MARKS               NaN  …
75            568               0           OUTPUT               NaN  …
76            334               0             FILE               NaN  …
77            631               0            PRINT               NaN  …

    study.partner.left.num  study.partner.right.num  \
73                     NaN                      NaN
74                     NaN                      NaN
75                   818.0                      NaN
76                   545.0                      NaN
77                   511.0                      NaN

    study.partner.left.string  study.partner.right.string  resp.num  \
73                        NaN                         NaN       NaN
74                        NaN                         NaN       NaN
75                   SUFFERED                         NaN       NaN
76                OBSERVATION                         NaN       NaN
77                    MESSAGE                         NaN       NaN

    study.partner.resp.num  study.partner.resp.string  recall.type  \
73                     NaN                        NaN          NaN
74                     NaN                        NaN          NaN
75                     NaN                        NaN          NaN
76                     NaN                        NaN          NaN
77                     NaN                        NaN          NaN

    recall.list       condition.block
```

```
73           NaN  Single recognition 1
74           NaN  Single recognition 1
75           NaN  Single recognition 1
76           NaN  Single recognition 1
77           NaN  Single recognition 1

[5 rows x 39 columns]
```

[7]:
```
# keeping the relevant columns for our study of single recognition
single_recognition_condition_original = single_recognition_condition.copy()
single_recognition_condition = single_recognition_condition[['block',␣
↪'subject', 'phase', 'condition', 'trial', 'stim.string.left',␣
↪'studied','resp', 'study.pos.left']]
single_recognition_condition.head()
```

[7]:
```
    block  subject phase           condition  trial stim.string.left  studied  \
73      5        2  test  Single recognition      1            PROVE        0
74      5        2  test  Single recognition      2            MARKS        0
75      5        2  test  Single recognition      3           OUTPUT        1
76      5        2  test  Single recognition      4             FILE        1
77      5        2  test  Single recognition      5            PRINT        1

    resp  study.pos.left
73     0              50
74     1              43
75     1              40
76     1               3
77     0               1
```

Single Recognition Human Accuracy

[8]:
```
# the people who responded that they have seen the object on screen with yes
# Yes = 1 and No = 0, only considering the trials which were for studying
single_recognition_condition =␣
↪single_recognition_condition[single_recognition_condition['studied'] == 1]
single_recongnition_human_acc = single_recognition_condition['resp'].sum()/␣
↪len(single_recognition_condition['resp'])
print(f"The human accuracy for single recognition cognitive test is: ␣
↪{round(single_recongnition_human_acc* 100,2)} %")
```

```
The human accuracy for single recognition cognitive test is:  83.51 %
```

[9]:
```
# single_acc_trial1 =␣
↪single_recognition_condition[single_recognition_condition['trial'] == 1]
single_acc_by_pos =␣
↪single_recognition_condition[single_recognition_condition['study.pos.
↪left']<21]
```

```
single_acc_by_pos = single_acc_by_pos.groupby('study.pos.left')[['resp',␣
↪'studied']].sum('resp')
single_acc_by_pos['prob'] = single_acc_by_pos['resp']/␣
↪single_acc_by_pos['studied']
single_acc_by_pos
```

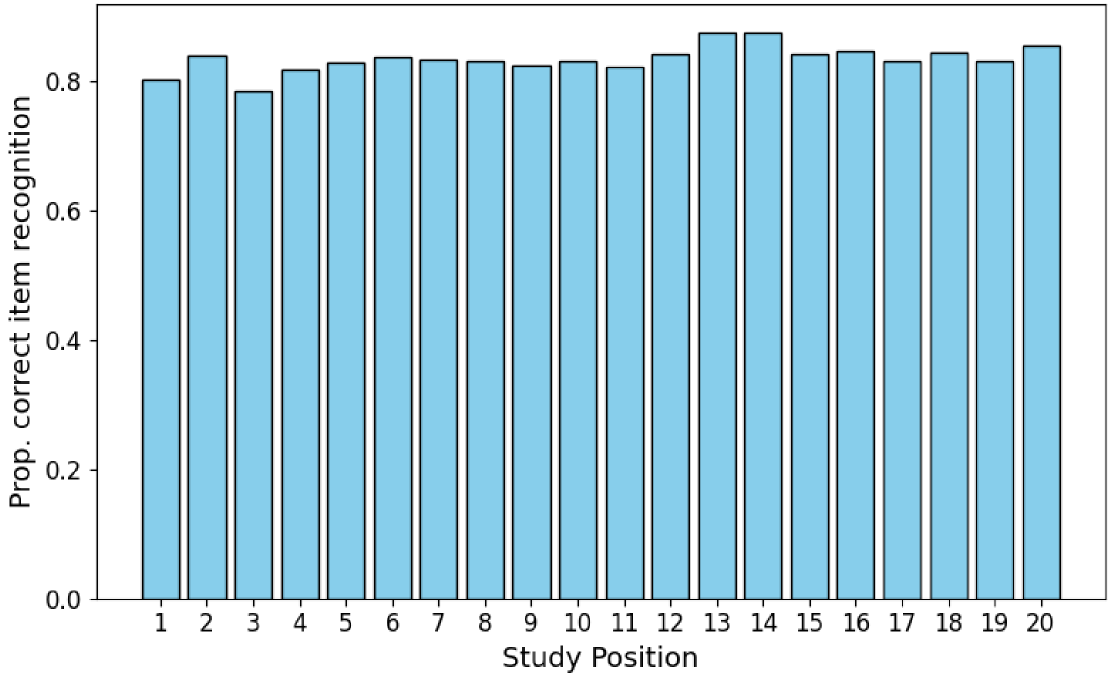
[9]:

study.pos.left	resp	studied	prob
1	281	351	0.800570
2	275	328	0.838415
3	266	339	0.784661
4	264	323	0.817337
5	259	313	0.827476
6	284	339	0.837758
7	267	321	0.831776
8	289	348	0.830460
9	258	313	0.824281
10	306	369	0.829268
11	275	335	0.820896
12	291	346	0.841040
13	299	342	0.874269
14	284	325	0.873846
15	284	338	0.840237
16	263	311	0.845659
17	286	345	0.828986
18	276	327	0.844037
19	286	345	0.828986
20	311	364	0.854396

[10]:

```
import matplotlib.pyplot as plt
def create_plot(x, y,  task_name, task_type):
  # Bar graph
  plt.figure(figsize=(8, 5))
  plt.bar(x,y, color='skyblue', edgecolor='black')

  # Add labels and title
  plt.xlabel('Study Position', fontsize=14)
  plt.ylabel(f'{task_type} correct {task_name}', fontsize=14)


  # Customize ticks
  plt.xticks(x,fontsize=12)
  plt.yticks(fontsize=12)

  # Show the plot
  plt.tight_layout()
  plt.show()
```

```
[11]: create_plot(x= single_acc_by_pos.index, y = single_acc_by_pos['prob'],␣
       ↪task_name= 'item recognition', task_type = 'Prop.')
```



0.1.6 2. Associative Recognition

```
[12]: associative_recognition_condition = data[data['condition'] == 'Associative␣
       ↪recognition']
      associative_recognition_condition.head()
```

```
[12]:    Unnamed: 0  subject  block phase               condition  trial  \
      0        37.0        2      1  test  Associative recognition      1
      1        38.0        2      1  test  Associative recognition      2
      2        39.0        2      1  test  Associative recognition      3
      3        40.0        2      1  test  Associative recognition      4
      4        41.0        2      1  test  Associative recognition      5

         stim.num.left  stim.num.right stim.string.left stim.string.right  …  \
      0            141             722            CLAIM         SCATTERED  …
      1            809             748          SHARPLY            STRESS  …
      2            894             352           FORMER            VOLUME  …
      3            683             189         CONTROLS         REMAINING  …
      4            857              30            TRACK        APPARENTLY  …

         study.partner.left.num  study.partner.right.num  study.partner.left.string  \
```

```
0                    722.0                      141.0                         SCATTERED
1                    747.0                      620.0                            SHARED
2                    352.0                      894.0                            FORMER
3                    919.0                      737.0                           WRAPPED
4                    665.0                      122.0                            READER

   study.partner.right.string  resp.num  study.partner.resp.num  \
0                       CLAIM       NaN                     NaN
1                      POURED       NaN                     NaN
2                      VOLUME       NaN                     NaN
3                   SENTENCES       NaN                     NaN
4                        CASH       NaN                     NaN

   study.partner.resp.string  recall.type  recall.list  \
0                        NaN          NaN          NaN
1                        NaN          NaN          NaN
2                        NaN          NaN          NaN
3                        NaN          NaN          NaN
4                        NaN          NaN          NaN

             condition.block
0  Associative recognition 1
1  Associative recognition 1
2  Associative recognition 1
3  Associative recognition 1
4  Associative recognition 1

[5 rows x 39 columns]
```

[13]:

```python
# keeping the relevant columns for our study of associative recognition:
associative_recognition_condition_original = associative_recognition_condition.
 ↪copy()
associative_recognition_condition = associative_recognition_condition[['block',␣
 ↪'subject', 'phase', 'condition', 'trial', 'stim.string.left', 'stim.string.
 ↪right', 'studied','resp', 'study.pos.left']]
associative_recognition_condition.head()
```

[13]:

```
   block  subject phase                condition  trial stim.string.left  \
0      1        2  test  Associative recognition      1            CLAIM
1      1        2  test  Associative recognition      2          SHARPLY
2      1        2  test  Associative recognition      3           FORMER
3      1        2  test  Associative recognition      4         CONTROLS
4      1        2  test  Associative recognition      5            TRACK

  stim.string.right  studied  resp  study.pos.left
0         SCATTERED        1     1              15
1            STRESS        0     1              18
```

```
2          VOLUME        1      1             20
3       REMAINING        0      1              5
4      APPARENTLY        0      1              3
```

Associative Recognition Human Accuracy

[14]:
```
#indicate if the PAIR of words you see on the screen was studied as a PAIR on␣
↪the list you just studied (YES) or were not a pair (NO), accuracy
associative_recognition_condition =␣
↪associative_recognition_condition[associative_recognition_condition['studied']␣
↪== 1]
associative_recongnition_human_acc = associative_recognition_condition['resp'].
↪sum()/ len(associative_recognition_condition['resp'])
print(f"The human accuracy for Associative recognition cognitive test is: ␣
↪{round(associative_recongnition_human_acc* 100,2)} %")
```

```
The human accuracy for Associative recognition cognitive test is:  80.34 %
```

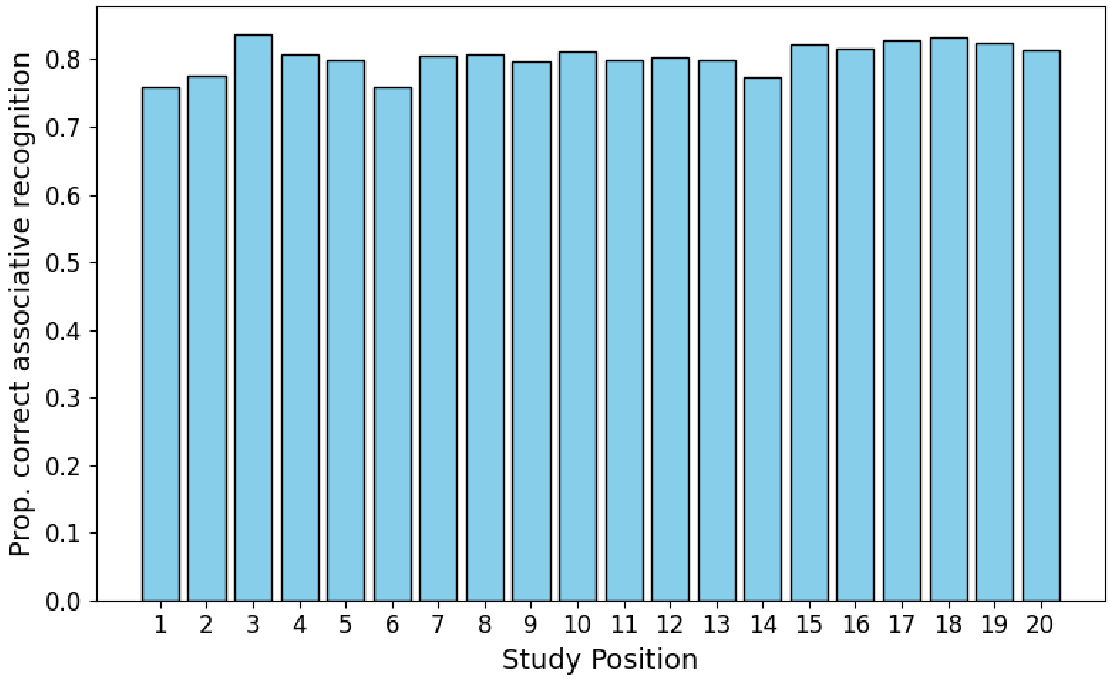
[15]:
```
associative_acc_by_pos =␣
↪associative_recognition_condition[associative_recognition_condition['study.
↪pos.left']<21]
associative_acc_by_pos = associative_acc_by_pos.groupby('study.pos.
↪left')[['resp', 'studied']].sum('resp')
associative_acc_by_pos['prob'] = associative_acc_by_pos['resp']/␣
↪associative_acc_by_pos['studied']
associative_acc_by_pos
```

[15]:

study.pos.left	resp	studied	prob
1	509	671	0.758569
2	534	689	0.775036
3	544	650	0.836923
4	529	656	0.806402
5	536	671	0.798808
6	502	661	0.759455
7	552	685	0.805839
8	540	669	0.807175
9	542	680	0.797059
10	551	679	0.811487
11	527	660	0.798485
12	528	658	0.802432
13	550	688	0.799419
14	519	670	0.774627
15	546	665	0.821053
16	564	691	0.816208
17	574	693	0.828283
18	571	686	0.832362

```
19                 523      634   0.824921
20                 541      665   0.813534
```

[16]:
```
create_plot(x= associative_acc_by_pos.index, y =␣
↪associative_acc_by_pos['prob'], task_name= 'associative recognition',␣
↪task_type = 'Prop.')
```



0.1.7 3. Cued Recall

[17]:
```
cued_recall_condition = data[data['condition'] == 'Cued recall']
cued_recall_condition.head()
```

[17]:
```
    Unnamed: 0  subject  block phase    condition  trial  stim.num.left  \
33       142.0        2      3  test  Cued recall      1            339
34       143.0        2      3  test  Cued recall      2            282
35       144.0        2      3  test  Cued recall      3            739
36       145.0        2      3  test  Cued recall      4            576
37       146.0        2      3  test  Cued recall      5            804

    stim.num.right stim.string.left stim.string.right  …  \
33               0           FIRMLY               NaN  …
34               0          ENJOYED               NaN  …
35               0        SERIOUSLY               NaN  …
36               0          PARTIES               NaN  …
```

```
37                0            STOCK             NaN  …

    study.partner.left.num  study.partner.right.num  \
33                   303.0                      NaN
34                   269.0                      NaN
35                   278.0                      NaN
36                    74.0                      NaN
37                   396.0                      NaN

    study.partner.left.string  study.partner.right.string  resp.num  \
33                  EXPANSION                         NaN       NaN
34                EFFECTIVELY                         NaN       NaN
35                  EMPLOYEES                         NaN       NaN
36                    BEDROOM                         NaN      74.0
37                      HAVEN                         NaN       NaN

    study.partner.resp.num  study.partner.resp.string          recall.type  \
33                     NaN                        NaN                  NaN
34                     NaN                        NaN                  NaN
35                     NaN                        NaN  Extralist intrusion
36                   576.0                    PARTIES              Correct
37                     NaN                        NaN                  NaN

    recall.list  condition.block
33          NaN   Cued recall 1
34          NaN   Cued recall 1
35          0.0   Cued recall 1
36          3.0   Cued recall 1
37          NaN   Cued recall 1

[5 rows x 39 columns]
```

[18]:
```python
# keeping the relevant columns for our study of cued recall:
cued_recall_condition_original = cued_recall_condition.copy()
cued_recall_condition = cued_recall_condition[['block', 'subject', 'phase',
 'condition', 'trial', 'stim.string.left', 'stim.string.right',
 'studied','resp.string', 'recall.type', 'study.pos.left']]
cued_recall_condition.head()
```

[18]:
```
    block  subject phase    condition  trial stim.string.left  \
33      3        2  test  Cued recall      1           FIRMLY
34      3        2  test  Cued recall      2          ENJOYED
35      3        2  test  Cued recall      3        SERIOUSLY
36      3        2  test  Cued recall      4          PARTIES
37      3        2  test  Cued recall      5            STOCK

   stim.string.right  studied resp.string          recall.type  study.pos.left
```

33	NaN	1	NaN	NaN	15
34	NaN	1	NaN	NaN	1
35	NaN	0	tough	Extralist intrusion	7
36	NaN	1	bedroom	Correct	2
37	NaN	1	NaN	NaN	13

Human Accuracy for Cued Recall

[19]:
```
# typing the OTHER WORD in the pair. For example if you studied BRICK BRACK and␣
↪you now see BRICK your response should be BRACK. If you cannot recall the␣
↪word click DON'T REMEMBER
cued_recall_condition = cued_recall_condition[(cued_recall_condition['studied']␣
↪== 1)]
cued_recall_human_acc = len(cued_recall_condition[cued_recall_condition['recall.
↪type'] == 'Correct'])/ len(cued_recall_condition)
print(f"The human accuracy for Cued Recall cognitive test is: ␣
↪{round(cued_recall_human_acc* 100,2)} %")
```

```
The human accuracy for Cued Recall cognitive test is:  31.6 %
```

[20]:
```
# Group by 'pos' and calculate the number of correct responses per trial
cued_acc_by_pos = cued_recall_condition.groupby('study.pos.left').apply(
    lambda group: (group['recall.type'] == 'Correct').sum()
).reset_index(name='prob')

# Display the result
cued_acc_by_pos
```

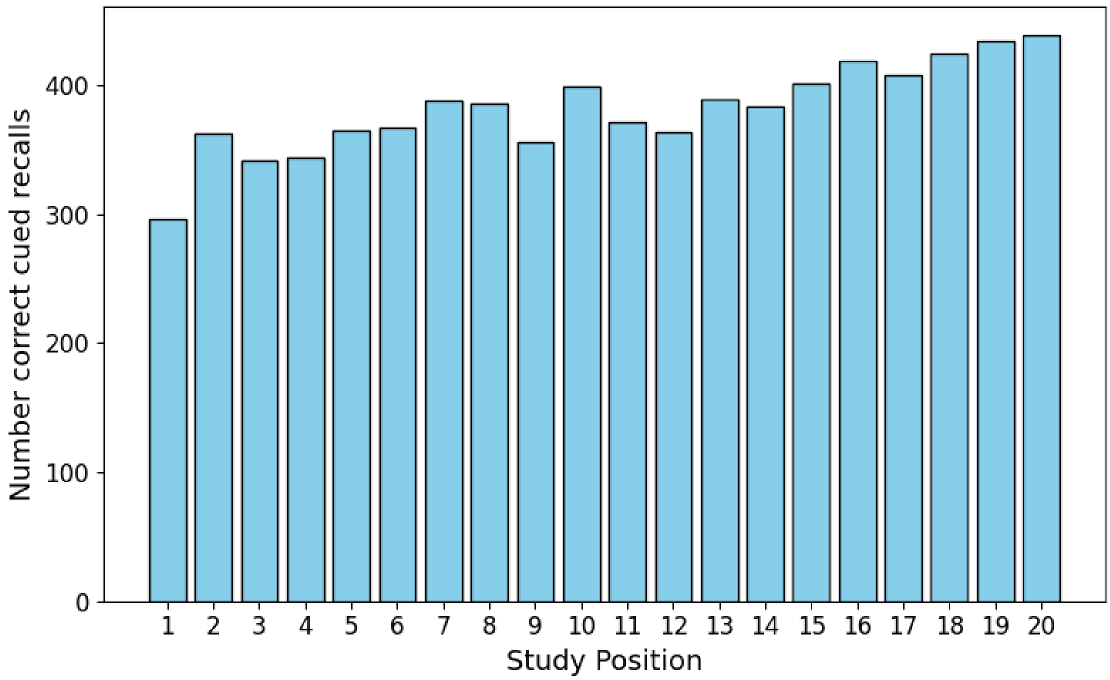
```
<ipython-input-20-de85434c4686>:2: DeprecationWarning: DataFrameGroupBy.apply
operated on the grouping columns. This behavior is deprecated, and in a future
version of pandas the grouping columns will be excluded from the operation.
Either pass `include_groups=False` to exclude the groupings or explicitly select
the grouping columns after groupby to silence this warning.
  cued_acc_by_pos = cued_recall_condition.groupby('study.pos.left').apply(
```

[20]:

	study.pos.left	prob
0	1	297
1	2	363
2	3	342
3	4	344
4	5	365
5	6	367
6	7	388
7	8	386
8	9	356
9	10	399
10	11	372

```
11                 12   364
12                 13   389
13                 14   384
14                 15   401
15                 16   419
16                 17   408
17                 18   424
18                 19   434
19                 20   439
```

[21]:
```python
create_plot(x= cued_acc_by_pos['study.pos.left'], y = cued_acc_by_pos['prob'],
 ↪task_name= 'cued recalls', task_type = 'Number')
```



0.1.8 4. Free Recall

[22]:
```python
free_recall_condition = data[data['condition'] == 'Free recall']
free_recall_condition.head()
```

[22]:
```
    Unnamed: 0  subject  block phase    condition  trial  stim.num.left  \
20        93.0        2      2  test  Free recall      1              0
21        94.0        2      2  test  Free recall      2              0
22        95.0        2      2  test  Free recall      3              0
23        96.0        2      2  test  Free recall      4              0
24        97.0        2      2  test  Free recall      5              0
```

```
    stim.num.right stim.string.left stim.string.right  …  \
20               0              NaN               NaN  …
21               0              NaN               NaN  …
22               0              NaN               NaN  …
23               0              NaN               NaN  …
24               0              NaN               NaN  …

    study.partner.left.num  study.partner.right.num  \
20                     NaN                      NaN
21                     NaN                      NaN
22                     NaN                      NaN
23                     NaN                      NaN
24                     NaN                      NaN

    study.partner.left.string  study.partner.right.string  resp.num  \
20                        NaN                         NaN     736.0
21                        NaN                         NaN       NaN
22                        NaN                         NaN       NaN
23                        NaN                         NaN       NaN
24                        NaN                         NaN     114.0

    study.partner.resp.num  study.partner.resp.string           recall.type  \
20                   126.0                    CENTERS               Correct
21                     NaN                        NaN                   NaN
22                     NaN                        NaN   Extralist intrusion
23                     NaN                        NaN   Extralist intrusion
24                     NaN                        NaN  Prior-list intrusion

    recall.list  condition.block
20          2.0    Free recall 1
21          NaN    Free recall 1
22          0.0    Free recall 1
23          0.0    Free recall 1
24          1.0    Free recall 1

[5 rows x 39 columns]
```

[23]:
```
# keeping the relevant columns for our study of free recall:
free_recall_condition_original = free_recall_condition.copy()
free_recall_condition = free_recall_condition[['block', 'subject', 'phase',
 ↪'condition', 'trial',  'studied','resp.string', 'resp.string.corr',
                                               'recall.type', 'study.pos.left',
 ↪'recall.list']]
free_recall_condition.head()
```

[23]:
```
    block  subject phase    condition  trial  studied      resp.string  \
20      2        2  test  Free recall      1        1           senate
21      2        2  test  Free recall      2        1            fiexd
22      2        2  test  Free recall      3        0            fixed
23      2        2  test  Free recall      4        0  characteristics
24      2        2  test  Free recall      5        0             camp

   resp.string.corr            recall.type  study.pos.left  recall.list
20           senate                Correct              27          2.0
21            fixed                    NaN               0          NaN
22            fixed    Extralist intrusion               0          0.0
23  characteristics    Extralist intrusion               0          0.0
24             camp  Prior-list intrusion                0          1.0
```

Human Accuracy for Free Recall

[24]:
```python
# "try to recall as many words from the study list as you possibly can. When␣
↪you cannot recall any more words click on the FINISHED button"
free_recall_condition = free_recall_condition[free_recall_condition['studied']␣
↪== 1]
free_recall_human_acc = len(free_recall_condition[free_recall_condition['recall.
↪type'] == 'Correct'])/ len(free_recall_condition)
print(f"The human accuracy for Free Recall cognitive test is: ␣
↪{round(free_recall_human_acc* 100,2)} %")
```

```
The human accuracy for Free Recall cognitive test is:  7.79 %
```

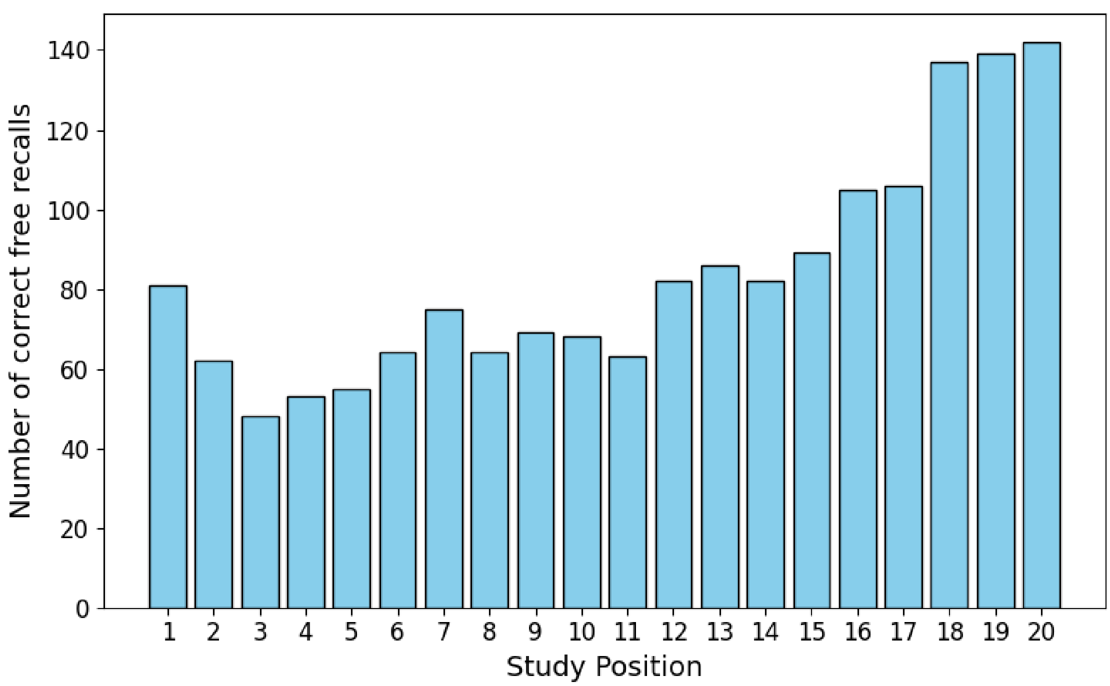
[25]:
```python
# Filter out positions within the valid range
free_acc_by_pos = free_recall_condition[(free_recall_condition['study.pos.
↪left'] > 0) & (free_recall_condition['study.pos.left'] < 21)]

# Group by study position and count correct recalls
free_acc_by_pos = free_acc_by_pos.groupby('study.pos.left').apply(
    lambda group: (group['recall.type'] == 'Correct').sum()
).reset_index(name='prob')

free_acc_by_pos
```

```
<ipython-input-25-785cf3a43480>:5: DeprecationWarning: DataFrameGroupBy.apply
operated on the grouping columns. This behavior is deprecated, and in a future
version of pandas the grouping columns will be excluded from the operation.
Either pass `include_groups=False` to exclude the groupings or explicitly select
the grouping columns after groupby to silence this warning.
  free_acc_by_pos = free_acc_by_pos.groupby('study.pos.left').apply(
```

[25]:
```
   study.pos.left  prob
0               1    81
```

1	2	62
2	3	48
3	4	53
4	5	55
5	6	64
6	7	75
7	8	64
8	9	69
9	10	68
10	11	63
11	12	82
12	13	86
13	14	82
14	15	89
15	16	105
16	17	106
17	18	137
18	19	139
19	20	142

[26]:
```
create_plot(x= free_acc_by_pos['study.pos.left'], y = free_acc_by_pos['prob'], task_name= 'free recalls', task_type = 'Number of')
```



0.1.9 5. Lexical Decision

[27]:
```
lexical_decision_condition = data[data['condition'] == 'Lexical decision']
lexical_decision_condition.head()
```

[27]:
```
    Unnamed: 0  subject  block phase         condition  trial  stim.num.left  \
53       162.0        2      4  test  Lexical decision      1             29
54       163.0        2      4  test  Lexical decision      2            514
55       164.0        2      4  test  Lexical decision      3            179
56       165.0        2      4  test  Lexical decision      4            519
57       166.0        2      4  test  Lexical decision      5            104

    stim.num.right stim.string.left stim.string.right  …  \
53               0        APARTMENT               NaN  …
54               0          RASSING               NaN  …
55               0       CONWRESHLY               NaN  …
56               0          MISSING               NaN  …
57               0         CRITHERS               NaN  …

    study.partner.left.num  study.partner.right.num  \
53                     NaN                      NaN
54                     NaN                      NaN
55                     NaN                      NaN
56                     NaN                      NaN
57                     NaN                      NaN

    study.partner.left.string  study.partner.right.string  resp.num  \
53                        NaN                         NaN       NaN
54                        NaN                         NaN       NaN
55                        NaN                         NaN       NaN
56                        NaN                         NaN       NaN
57                        NaN                         NaN       NaN

    study.partner.resp.num  study.partner.resp.string  recall.type  \
53                     NaN                        NaN          NaN
54                     NaN                        NaN          NaN
55                     NaN                        NaN          NaN
56                     NaN                        NaN          NaN
57                     NaN                        NaN          NaN

    recall.list     condition.block
53          NaN  Lexical decision 1
54          NaN  Lexical decision 1
55          NaN  Lexical decision 1
56          NaN  Lexical decision 1
57          NaN  Lexical decision 1
```

```
[5 rows x 39 columns]
```

[28]:
```python
lexical_decision_condition_original = lexical_decision_condition.copy()
lexical_decision_condition = lexical_decision_condition[['block', 'subject',␣
↪'phase', 'condition', 'trial', 'stim.string.left',  'studied', 'resp']]
lexical_decision_condition.head()
```

[28]:
```
    block  subject phase          condition  trial stim.string.left  studied  \
53      4        2  test  Lexical decision      1        APARTMENT        1
54      4        2  test  Lexical decision      2          RASSING        0
55      4        2  test  Lexical decision      3       CONWRESHLY        0
56      4        2  test  Lexical decision      4          MISSING        1
57      4        2  test  Lexical decision      5         CRITHERS        0

    resp
53     1
54     0
55     0
56     1
57     0
```

[29]:
```python
# "indicate if the item you see is a word (YES) or not at word (NO). Respond as␣
↪QUICKLY as possible
lexical_decision_condition =␣
↪lexical_decision_condition[lexical_decision_condition['studied'] == 1]
lexical_decision_human_acc = lexical_decision_condition['resp'].sum()/␣
↪len(lexical_decision_condition['resp'])
print(f"The human accuracy for Lexical Decision cognitive test is: ␣
↪{round(lexical_decision_human_acc* 100,2)} %")
```

```
The human accuracy for Lexical Decision cognitive test is:  95.78 %
```

0.2 LLM Modelling

[30]:
```python
from transformers import T5Tokenizer, T5ForConditionalGeneration
import torch

model_name="google/flan-t5-large"
tokenizer = T5Tokenizer.from_pretrained(model_name)
model = T5ForConditionalGeneration.from_pretrained(model_name,␣
↪device_map="auto")

# Check if GPU is available
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
model.to(device)
```

```
/usr/local/lib/python3.10/dist-packages/huggingface_hub/utils/_auth.py:94:
```

```
UserWarning:
The secret `HF_TOKEN` does not exist in your Colab secrets.
To authenticate with the Hugging Face Hub, create a token in your settings tab
(https://huggingface.co/settings/tokens), set it as secret in your Google Colab
and restart your session.
You will be able to reuse this secret in all of your notebooks.
Please note that authentication is recommended but still optional to access
public models or datasets.
  warnings.warn(
```

```
tokenizer_config.json:   0%|          | 0.00/2.54k [00:00<?, ?B/s]
```

```
spiece.model:   0%|          | 0.00/792k [00:00<?, ?B/s]
```

```
special_tokens_map.json:   0%|          | 0.00/2.20k [00:00<?, ?B/s]
```

```
tokenizer.json:   0%|          | 0.00/2.42M [00:00<?, ?B/s]
```

```
You are using the default legacy behaviour of the <class
'transformers.models.t5.tokenization_t5.T5Tokenizer'>. This is expected, and
simply means that the `legacy` (previous) behavior will be used so nothing
changes for you. If you want to use the new behaviour, set `legacy=False`. This
should only be set if you understand what it means, and thoroughly read the
reason why this was added as explained in
https://github.com/huggingface/transformers/pull/24565
```

```
config.json:   0%|          | 0.00/662 [00:00<?, ?B/s]
```

```
model.safetensors:   0%|          | 0.00/3.13G [00:00<?, ?B/s]
```

```
generation_config.json:   0%|          | 0.00/147 [00:00<?, ?B/s]
```

[30]:
```
T5ForConditionalGeneration(
  (shared): Embedding(32128, 1024)
  (encoder): T5Stack(
    (embed_tokens): Embedding(32128, 1024)
    (block): ModuleList(
      (0): T5Block(
        (layer): ModuleList(
          (0): T5LayerSelfAttention(
            (SelfAttention): T5Attention(
              (q): Linear(in_features=1024, out_features=1024, bias=False)
              (k): Linear(in_features=1024, out_features=1024, bias=False)
              (v): Linear(in_features=1024, out_features=1024, bias=False)
              (o): Linear(in_features=1024, out_features=1024, bias=False)
              (relative_attention_bias): Embedding(32, 16)
            )
            (layer_norm): T5LayerNorm()
            (dropout): Dropout(p=0.1, inplace=False)
          )
          (1): T5LayerFF(
```

```
          (DenseReluDense): T5DenseGatedActDense(
            (wi_0): Linear(in_features=1024, out_features=2816, bias=False)
            (wi_1): Linear(in_features=1024, out_features=2816, bias=False)
            (wo): Linear(in_features=2816, out_features=1024, bias=False)
            (dropout): Dropout(p=0.1, inplace=False)
            (act): NewGELUActivation()
          )
          (layer_norm): T5LayerNorm()
          (dropout): Dropout(p=0.1, inplace=False)
        )
      )
    )
    (1-23): 23 x T5Block(
      (layer): ModuleList(
        (0): T5LayerSelfAttention(
          (SelfAttention): T5Attention(
            (q): Linear(in_features=1024, out_features=1024, bias=False)
            (k): Linear(in_features=1024, out_features=1024, bias=False)
            (v): Linear(in_features=1024, out_features=1024, bias=False)
            (o): Linear(in_features=1024, out_features=1024, bias=False)
          )
          (layer_norm): T5LayerNorm()
          (dropout): Dropout(p=0.1, inplace=False)
        )
        (1): T5LayerFF(
          (DenseReluDense): T5DenseGatedActDense(
            (wi_0): Linear(in_features=1024, out_features=2816, bias=False)
            (wi_1): Linear(in_features=1024, out_features=2816, bias=False)
            (wo): Linear(in_features=2816, out_features=1024, bias=False)
            (dropout): Dropout(p=0.1, inplace=False)
            (act): NewGELUActivation()
          )
          (layer_norm): T5LayerNorm()
          (dropout): Dropout(p=0.1, inplace=False)
        )
      )
    )
  )
  (final_layer_norm): T5LayerNorm()
  (dropout): Dropout(p=0.1, inplace=False)
)
(decoder): T5Stack(
  (embed_tokens): Embedding(32128, 1024)
  (block): ModuleList(
    (0): T5Block(
      (layer): ModuleList(
        (0): T5LayerSelfAttention(
```

```
      (SelfAttention): T5Attention(
        (q): Linear(in_features=1024, out_features=1024, bias=False)
        (k): Linear(in_features=1024, out_features=1024, bias=False)
        (v): Linear(in_features=1024, out_features=1024, bias=False)
        (o): Linear(in_features=1024, out_features=1024, bias=False)
        (relative_attention_bias): Embedding(32, 16)
      )
      (layer_norm): T5LayerNorm()
      (dropout): Dropout(p=0.1, inplace=False)
    )
    (1): T5LayerCrossAttention(
      (EncDecAttention): T5Attention(
        (q): Linear(in_features=1024, out_features=1024, bias=False)
        (k): Linear(in_features=1024, out_features=1024, bias=False)
        (v): Linear(in_features=1024, out_features=1024, bias=False)
        (o): Linear(in_features=1024, out_features=1024, bias=False)
      )
      (layer_norm): T5LayerNorm()
      (dropout): Dropout(p=0.1, inplace=False)
    )
    (2): T5LayerFF(
      (DenseReluDense): T5DenseGatedActDense(
        (wi_0): Linear(in_features=1024, out_features=2816, bias=False)
        (wi_1): Linear(in_features=1024, out_features=2816, bias=False)
        (wo): Linear(in_features=2816, out_features=1024, bias=False)
        (dropout): Dropout(p=0.1, inplace=False)
        (act): NewGELUActivation()
      )
      (layer_norm): T5LayerNorm()
      (dropout): Dropout(p=0.1, inplace=False)
    )
  )
)
(1-23): 23 x T5Block(
  (layer): ModuleList(
    (0): T5LayerSelfAttention(
      (SelfAttention): T5Attention(
        (q): Linear(in_features=1024, out_features=1024, bias=False)
        (k): Linear(in_features=1024, out_features=1024, bias=False)
        (v): Linear(in_features=1024, out_features=1024, bias=False)
        (o): Linear(in_features=1024, out_features=1024, bias=False)
      )
      (layer_norm): T5LayerNorm()
      (dropout): Dropout(p=0.1, inplace=False)
    )
    (1): T5LayerCrossAttention(
      (EncDecAttention): T5Attention(
```

```
                (q): Linear(in_features=1024, out_features=1024, bias=False)
                (k): Linear(in_features=1024, out_features=1024, bias=False)
                (v): Linear(in_features=1024, out_features=1024, bias=False)
                (o): Linear(in_features=1024, out_features=1024, bias=False)
              )
              (layer_norm): T5LayerNorm()
              (dropout): Dropout(p=0.1, inplace=False)
            )
            (2): T5LayerFF(
              (DenseReluDense): T5DenseGatedActDense(
                (wi_0): Linear(in_features=1024, out_features=2816, bias=False)
                (wi_1): Linear(in_features=1024, out_features=2816, bias=False)
                (wo): Linear(in_features=2816, out_features=1024, bias=False)
                (dropout): Dropout(p=0.1, inplace=False)
                (act): NewGELUActivation()
              )
              (layer_norm): T5LayerNorm()
              (dropout): Dropout(p=0.1, inplace=False)
            )
          )
        )
      )
      (final_layer_norm): T5LayerNorm()
      (dropout): Dropout(p=0.1, inplace=False)
    )
    (lm_head): Linear(in_features=1024, out_features=32128, bias=False)
  )
```

Creating Function for Study Phase

[31]:
```
import random

study_list_associative_left =␣
↪list(associative_recognition_condition_original['stim.string.left'])
study_list_associative_right =␣
↪list(associative_recognition_condition_original['stim.string.right'])


def create_word_pairs(left_words, right_words):
    """
    Creates a list of word pairs by combining corresponding elements from
    two separate lists.

    Args:
      left_words: List of left words.
      right_words: List of right words.
```

```
    Returns:
      A list of word pairs, where each pair is a tuple of two words.
    """
    if len(left_words) != len(right_words):
      raise ValueError("Left and right word lists must have the same length.")

    return list(zip(left_words, right_words))

study_list_final = create_word_pairs(study_list_associative_left,
 ↪study_list_associative_right)

# Context management: Store study lists for each subject and block
context = {}

# Function to add a study list to the context
def add_study_list(block_id, study_list):
    if block_id not in context:
       context[block_id] = {"study_list": study_list, "tasks": []}


def get_separated_words(study_list_input):
  return  [word for tup in study_list_input for word in tup]

words_list_foil = [
    'PROVE', 'MARKS', 'OUTPUT', 'FILE', 'PRINT', 'SIZES', 'TUBE', 'FEARED',
    'POLLUTION', 'DESIGN', 'OPPORTUNITIES', 'ESCAPE', 'STRING', 'STANDS',
    'HAT', 'SLIPPED', 'OCCUPIED', 'HARMFUL', 'SPLIT', 'SPOKEN', 'ATTACHED',
    'RESPOND', 'GUN', 'BLEW', 'STORM', 'FINGER', 'PROPERTIES', 'IMPACT',
    'BURIED', 'TESTS']


def scramble_word_pairs(word_pairs):
  """
  Scrambles the word pairs by randomly rearranging them.

  Args:
    word_pairs: List of word pairs.

  Returns:
    A list of scrambled word pairs.
  """
  scrambled_pairs = word_pairs.copy()
  random.shuffle(scrambled_pairs)
  for pair in scrambled_pairs:
    random.shuffle(list(pair))  # Shuffle the words within each pair
  return scrambled_pairs
```

```python
def create_cued_recall_test(word_pairs):
  """
  Creates a list of cues for the cued recall test.

  Args:
    word_pairs: List of word pairs.

  Returns:
    A list of cues, where each cue is one word from a pair.
  """
  cues = []
  for x,y in word_pairs:
    # Randomly select one word from each pair as the cue
    cue = x
    cues.append(cue)
  random.shuffle(cues)  # Shuffle the order of cues
  return cues
```

Creating Contextual Prompt

[32]:
```python
# Function to construct a prompt with retained context
def construct_contextual_prompt(task_name, block_id, task_details):
    study_list = context[block_id]["study_list"]
    previous_tasks = context[block_id]["tasks"]
    study_list_str = ""

    for first_word, second_word in study_list:
      study_list_str += f"{first_word} - {second_word}, "

    previous_tasks_str = ". ".join(previous_tasks)

    # Combine context and task-specific details into a prompt
    prompt = (
        f"You are a memory assistant.\n"
        f"Previous Tasks:\n{previous_tasks_str}\n\n"
        f"Study Phase:\nYou studied the following word pairs:
 ↪\n{study_list_str}\n\n"
        f"Task Instructions:\nFocus only on the following task. Do not use␣
 ↪information from previous tasks.\n"
        f"Current Task: {task_name}\n{task_details}\n"
    )
    return prompt
```

[33]:
```python
def generate_study_list_for_position(position, list_length=30):
    """
```

```
    Generate a study list where a specific word is highlighted at a given␣
↪position.
    Ensures the same list is generated for the same position.

    Args:
        position (int): The position in the study list to emphasize (1-based␣
↪index).
        list_length (int): Total number of words in the study list.
    Returns:
        list: A study list of words with the target word at the specified␣
↪position.
    """

    if list_length > len(study_list_final):
        raise ValueError("Study list length exceeds vocabulary size.")

    # Set a fixed random seed based on the position to ensure consistency
    random.seed(position)

    # Select words from the vocabulary
    selected_words = random.sample(study_list_final, list_length)

    return selected_words
```

0.2.1 1. Single Recognition Condition Prompt

```
[34]: def single_item_recognition(block_id, test_items):
    """
    Simulates the single-item recognition task.

    Args:
        block_id (int): The block identifier.
        test_items (list): Words to be tested (half old, half foil).

    Returns:
        list: Model responses.
    """
    prompts = []
    for word in test_items:
        prompt = construct_contextual_prompt(
            task_name="Single-Item Recognition",
            block_id=block_id,
            task_details=f"Check the study list carefully to ensure your␣
↪response is accurate.\n"
                         f"The word to verify is: '{word}'\n"
                         f"Question:\n"
```

```
                         f"Is the word '{word}' present in the list that you␣
↪have studied? Respond with 'YES' or 'NO'.\n"
                         f"Please verify your answer before responding"
        )
        prompts.append(prompt)

    # Generate LLM responses for each prompt
    responses = []
    for prompt in prompts:
        inputs = tokenizer(prompt, return_tensors="pt").to(device)
        output = model.generate(**inputs, max_new_tokens=100, do_sample=True,␣
↪temperature=0.6)
        response = tokenizer.decode(output[0], skip_special_tokens=True)
        responses.append(response.strip())

    # Log the task to context
    context[block_id]["tasks"].append("Single-Item Recognition")

    return responses


def calculate_accuracy(responses, test_list, target_list):
    """
    Calculates the accuracy of the LLM's memory test.

    Args:
        responses: List of booleans indicating whether the LLM correctly␣
↪identified
                   old items.
        test_list: List of words in the test phase.
        target_list: List of words in the study phase.

    Returns:
        Accuracy: A float representing the proportion of correct responses.
    """

    correct_hits = 0
    correct_word = 0
    correct_foil = 0

    for i, word in enumerate(test_list):
        if word in target_list and responses[i].upper() == "YES":
            correct_hits += 1
            correct_word += 1
        elif word not in target_list and responses[i].upper() == "NO":
            correct_hits += 1
            correct_foil += 1
```

```
    total_targets_in_test = sum(1 for word in test_list if word in target_list)
    total_foils_in_test = sum(1 for word in test_list if word not in␣
↪target_list)

    # Calculate probabilities
    p_target = correct_word / total_targets_in_test
    p_foil = correct_foil / total_foils_in_test
    accuracy = correct_hits / len(test_list)

    print("Correct Hits (Word present in Study List and detected correctly to␣
↪be old):", correct_word)
    print("Correct Rejections (Word not present in Study List and detected␣
↪correctly to be new):", correct_foil)
    print(f"Total Accuracy: {accuracy:.2f}, p_target: {p_target:.2f}, p_foil:␣
↪{p_foil:.2f}")

    return accuracy, p_target, p_foil
```

[35]:
```
def testing_single_item(block_id, study_list):

  # Add study list to context
  add_study_list(block_id, study_list)

  study_list_first = get_separated_words(study_list)

  # selecting 10 items from study list
  target_words_first = random.sample(study_list_first, 10)

  # Selecting 10 foils items from word list which are not present in study list
  selected_test_list = random.sample(words_list_foil, 10)
  test_list =  target_words_first + selected_test_list

  # Shuffling list
  random.shuffle(test_list)

  single_item_results = single_item_recognition(block_id, test_list)
  print("Single-Item Recognition Results:", single_item_results)

  accuracy_single, single_prob_target, single_prob_foil =␣
↪calculate_accuracy(single_item_results, test_list, target_words_first)

  return accuracy_single, single_prob_target, single_prob_foil
```

[36]:
```
study_data_1 = generate_study_list_for_position(position=1)
block_id = 1
```

```
target_means = []
foil_means = []
```

[60]:
```
accuracy_single_1, single_prob_target_1, single_prob_foil_1 =␣
↪testing_single_item(block_id= block_id, study_list = study_data_1)
target_means.append(single_prob_target_1)
foil_means.append(single_prob_foil_1)
```

```
Single-Item Recognition Results: ['YES', 'NO', 'YES', 'NO', 'YES', 'NO', 'No',
'NO', 'NO', 'NO', 'Yes', 'Yes', 'NO', 'Yes', 'Yes', 'Yes', 'NO', 'NO', 'YES',
'YES']
Correct Hits (Word present in Study List and detected correctly to be old): 6
Correct Rejections (Word not present in Study List and detected correctly to be
new): 6
Total Accuracy: 0.60, p_target: 0.60, p_foil: 0.60
```

0.2.2 2. Associative Recognition Condition Prompt

[38]:
```
# Example for Associative Recognition
def associative_recognition(block_id, test_pairs):
    """
    Simulates the associative recognition task.

    Args:
        block_id (int): The block identifier.
        test_pairs (list): Pairs to be tested (half intact, half scrambled).

    Returns:
        list: Model responses.
    """
    prompts = []
    for pair in test_pairs:
        word1, word2 = pair
        prompt = construct_contextual_prompt(
            task_name="Associative Recognition",
            block_id=block_id,
            task_details=f"Check the study list carefully to ensure your␣
↪response is accurate.\n"
            f"The pair to verify is: '{word1}'-'{word2}'\n"
            f"Question:\n"
            f"Is the pair '{word1}'-'{word2}' present in the list together that␣
↪you have studied? Respond with 'YES' or 'NO'.\n"
            f"Please verify your answer before responding"
        )
        prompts.append(prompt)

    # Generate LLM responses for each prompt
```

```
    responses = []
    for prompt in prompts:
        inputs = tokenizer(prompt, return_tensors="pt").to(device)
        output = model.generate(**inputs, max_new_tokens=100, do_sample=True,␣
 ↪temperature=0.6)
        response = tokenizer.decode(output[0], skip_special_tokens=True)
        responses.append(response.strip())

    # Log the task to context
    context[block_id]["tasks"].append("Associative Recognition")

    return responses


def calculate_accuracy_associative(response, test_pairs, target_pairs):
  """
  Calculates the accuracy of the LLM's associative recognition test.

  Args:
      responses: List of booleans indicating whether the LLM correctly␣
 ↪identified
                 old items.
      test_pairs: List of test word pairs.
      target_pairs: List of original word pairs.

  Returns:
      Accuracy: A float representing the proportion of correct responses.
  """

  correct_hits = 0
  correct_word = 0
  correct_foil = 0

  for i, pair in enumerate(test_pairs):
      if pair in target_pairs:
          if response[i].upper() == "YES":
              correct_hits += 1
              correct_word += 1
      else:  # Word is not in the study list
          if response[i].upper() == "NO":
              correct_hits += 1
              correct_foil += 1

  total_targets_in_test = sum(1 for pair in test_pairs if pair in target_pairs)
  total_foils_in_test = sum(1 for pair in test_pairs if pair not in␣
 ↪target_pairs)
  # Calculate probabilities
```

```
    p_target = correct_word / total_targets_in_test
    p_foil = correct_foil / total_foils_in_test

    # Calculate accuracy
    total_responses = len(test_pairs)
    accuracy = correct_hits / total_responses

    print("Correct Hits (Target pairs correctly identified as old):",␣
↪correct_word)
    print("Correct Rejections (Foil pairs correctly identified as new):",␣
↪correct_foil)
    print(f"Total Accuracy: {accuracy:.2f}, p_target: {p_target:.2f}, p_foil:␣
↪{p_foil:.2f}")

    return accuracy, p_target, p_foil
```

[39]:

```
def testing_assosciative_recog(block_id, study_list):

    add_study_list(block_id, study_list)

    # Testing the model accuracy
    scrambled_pairs_associative = scramble_word_pairs(study_list)

    word_pairs_list = random.sample(study_list, 10)
    selected_test_pair_list = random.sample(scrambled_pairs_associative, 10)
    test_pair_list =  word_pairs_list + selected_test_pair_list

    # Shuffling list
    random.shuffle(test_pair_list)

    print(f"The Study list consists of words-pair: {word_pairs_list}\n The␣
↪Test-Pair List consists of words {test_pair_list}")
    associative_results = associative_recognition(block_id, test_pair_list)
    accuracy_asso, associative_prob_target, associative_prob_foil =␣
↪calculate_accuracy_associative(response= associative_results, test_pairs=␣
↪test_pair_list, target_pairs= word_pairs_list)

    return accuracy_asso, associative_prob_target, associative_prob_foil
```

[64]:

```
accuracy_asso_1, associative_prob_target_1, associative_prob_foil_1 =␣
 ↪testing_assosciative_recog(block_id= 1, study_list= study_data_1)
target_means.append(associative_prob_target_1)
foil_means.append(associative_prob_foil_1)
```

```
The Study list consists of words-pair: [('JOURNEY', 'VALLEYS'), ('STAR', 'ILL'),
('RESPOND', 'CLOUD'), ('DISTANT', 'GASES'), ('HABIT', 'DISAPPEARED'), ('RODE',
'CHARACTERS'), ('PARENT', 'BRUSH'), ('UNIQUE', 'REALITY'), ('OCCURRED',
```

```
'NURSE'), ('PROVE', 'EXPERIENCES')]
 The Test-Pair List consists of words [('RODE', 'CHARACTERS'), ('OCCURRED',
'NURSE'), ('APPOINTED', 'LOOSE'), ('STAR', 'ILL'), ('DISTANT', 'GASES'),
('JOURNEY', 'VALLEYS'), ('UNIQUE', 'REALITY'), ('RESPOND', 'CLOUD'), ('DISTANT',
'GASES'), ('OCCURRED', 'NURSE'), ('LEADERSHIP', 'NEIGHBORHOOD'), ('BELIEF',
'SEX'), ('STAR', 'ILL'), ('HABIT', 'FORMING'), ('PROVE', 'EXPERIENCES'),
('RESPOND', 'CLOUD'), ('PARENT', 'BRUSH'), ('HABIT', 'DISAPPEARED'), ('HABIT',
'DISAPPEARED'), ('VOICES', 'LOVELY')]
Correct Hits (Target pairs correctly identified as old): 12
Correct Rejections (Foil pairs correctly identified as new): 1
Total Accuracy: 0.65, p_target: 0.80, p_foil: 0.20
```

0.2.3 3. Cued Recall Condition Prompt

[41]:
```python
def cued_recall(block_id, cue_words):
    """
    Simulates the cued recall task.

    Args:
        block_id (int): The block identifier.
        cue_words (list): Words used as cues from the study list.

    Returns:
        list: Model responses.
    """
    prompts = []
    for cue in cue_words:
        prompt = construct_contextual_prompt(
            task_name="Cued Recall",
            block_id=block_id,
            task_details=f"During the study phase, you saw a list of word pairs.
 ↪ If you see one word from a pair, respond with the other word from the pair.␣
 ↪If you cannot remember the word, respond with 'DON'T REMEMBER'. Do not␣
 ↪provide any additional responses or commentary.\n"
                         f"\nThe word is '{cue}'. What is the paired word with␣
 ↪it?"
        )
        prompts.append(prompt)

    responses = []
    for prompt in prompts:
        inputs = tokenizer(prompt, return_tensors="pt").to(device)
        output = model.generate(**inputs, max_new_tokens=100, do_sample=True,␣
 ↪temperature=0.6)
        response = tokenizer.decode(output[0], skip_special_tokens=True)
        responses.append(response.strip())
```

```
    # Log the task to context
    context[block_id]["tasks"].append("Cued Recall")
    return responses
```

[42]:
```
def testing_cued_recall(block_id, study_list):

  expected_responses_list = []

  # Add study list to context
  add_study_list(block_id, study_list)

  # Testing the model accuracy
  cues = create_cued_recall_test(study_list[:10])
  print("Cued words list: ", cues)

  for cue in cues:
    # Determine the expected response (the other word in the pair)
    expected_response = None
    for pair in study_list[:10]:
      if cue in pair:
        expected_response = pair[0] if cue == pair[1] else pair[1]
        expected_responses_list.append(expected_response)
        break

  responses = cued_recall(block_id=1, cue_words=cues)
  correct_responses = 0
  foil_responses = 0

  for i, response in enumerate(responses):
      if response == expected_responses_list[i]:
          correct_responses += 1
      elif response in [pair[0] for pair in study_list[:10]] + [pair[1] for␣
↪pair in study_list[:10]]:
          foil_responses += 1

  # Calculate probabilities
  p_target = correct_responses / len(cues)
  p_foil = foil_responses / len(cues)

  print(f"Probablity of Cued Recall: {p_target}")
  print(f"Probability of Foil Responses: {p_foil}")
  print(f"Accuracy of Cued Recall: {p_target}")

  return p_target, p_foil
```

[43]:
```
p_target_cued, p_foil_cued = testing_cued_recall(block_id = 1, study_list =␣
↪study_data_1)
```

```
target_means.append(p_target_cued)
foil_means.append(p_foil_cued)
```

```
Cued words list:  ['BUYING', 'RODE', 'EXCHANGE', 'PROVE', 'SPECIALIZED',
'CHOSE', 'CABIN', 'BELIEF', 'SECONDS', 'VITAL']
Probablity of Cued Recall: 0.7
Probability of Foil Responses: 0.1
Accuracy of Cued Recall: 0.7
```

0.2.4 4. Free Recall Condition Prompt

[44]:
```
def free_recall(block_id):
    """
    Simulates the free recall task.

    Args:
        block_id (int): The block identifier.

    Returns:
        str: Model's attempt to recall as many words as possible from the study␣
↪list.
    """
    prompts = []
    prompt = construct_contextual_prompt(
        task_name="Free Recall",
        block_id=block_id,
        task_details="Your task is to recall as many words from the study list␣
↪as possible.\n"
                     "Write the words as a single comma-separated list,␣
↪replacing - with ,. For example, if you recall 'CLAIM - SCATTERED,' write it␣
↪as CLAIM, SCATTERED.\n"
                     "When you cannot recall any more words, type 'FINISHED'.\n"
                     "Do not respond immediately with 'FINISHED' without first␣
↪attempting to recall."
    )
    prompts.append(prompt)

    # Generate LLM response for free recall
    responses = []
    for prompt in prompts:
      inputs = tokenizer(prompt, return_tensors="pt").to(device)
      output = model.generate(**inputs, max_new_tokens=100, do_sample=True,␣
↪temperature=0.6)
      response = tokenizer.decode(output[0], skip_special_tokens=True)
      responses.append(response)
    recalled_words = set(response.upper().split(","))
```

```python
    recalled_words = set([word.strip() for word in recalled_words])

    # Log the task to context
    context[block_id]["tasks"].append("Free Recall")

    return response.strip()
```

[45]:
```python
def testing_free_recall(block_id, study_list):
  separated_list = get_separated_words(study_list)
  add_study_list(block_id, study_list)
  recalled_words = free_recall(block_id)
  p_target = 0
  p_foil = 0
  correct_remember = 0
  foil_remember = 0

  recalled_words = recalled_words.split(",")
  correct_remember = sum(1 for word in recalled_words if word.strip() in␣
 ↪separated_list)
  foil_remember = sum(1 for word in recalled_words if word.strip() not in␣
 ↪separated_list)

  total_recalled = len(recalled_words)
  total_study_words = len(separated_list)

  # Handle edge cases where no words are recalled
  p_target = correct_remember / total_study_words
  p_foil = foil_remember / total_recalled
  recall_score = correct_remember / total_recalled

  print(f"Probablity got for target for block {block_id}: ", p_target)
  print(f"Probablity got for foil for block {block_id}: ", p_foil)
  print(f"Accuracy got for free recall for block {block_id}: ", recall_score)

  return recall_score, p_target, p_foil, correct_remember
```

[46]:
```python
accuracy_free_recall, p_target_free, p_foil_free, correct_remember =␣
 ↪testing_free_recall(block_id = 1, study_list = study_data_1)
target_means.append(p_target_free)
foil_means.append(p_foil_free)
```

```
Probablity got for target for block 1:  0.4
Probablity got for foil for block 1:  0.04
Accuracy got for free recall for block 1:  0.96
```

0.2.5 Figure 18 comparison of free recall for both LLM and Human

[47]:
```
import pandas as pd

# Initialize variables
study_positions = list(range(1, 21))
results = []

# Perform multiple iterations for smoothing
iterations = 2

for position in study_positions:
    total_correct_remember = []
    for _ in range(iterations):
        # Generate a study list for the current position
        study_list = generate_study_list_for_position(position)
        _, _, _, correct_remember = testing_free_recall(block_id=position,
 ↪study_list=study_list)
        total_correct_remember.append(correct_remember)

    results.append({'study.pos.left': position, 'num_corr':
 ↪sum(total_correct_remember)})

# Convert results to a DataFrame
model_acc_by_pos = pd.DataFrame(results)
model_acc_by_pos
```

```
Probablity got for target for block 1:  0.38333333333333336
Probablity got for foil for block 1:  0.041666666666666664
Accuracy got for free recall for block 1:  0.9583333333333334
Probablity got for target for block 1:  0.4
Probablity got for foil for block 1:  0.04
Accuracy got for free recall for block 1:  0.96
Probablity got for target for block 2:  0.3333333333333333
Probablity got for foil for block 2:  0.047619047619047616
Accuracy got for free recall for block 2:  0.9523809523809523
Probablity got for target for block 2:  0.31666666666666665
Probablity got for foil for block 2:  0.09523809523809523
Accuracy got for free recall for block 2:  0.9047619047619048
Probablity got for target for block 3:  0.25
Probablity got for foil for block 3:  0.16666666666666666
Accuracy got for free recall for block 3:  0.8333333333333334
Probablity got for target for block 3:  0.16666666666666666
Probablity got for foil for block 3:  0.23076923076923078
Accuracy got for free recall for block 3:  0.7692307692307693
Probablity got for target for block 4:  0.38333333333333336
Probablity got for foil for block 4:  0.041666666666666664
Accuracy got for free recall for block 4:  0.9583333333333334
```

```
Probablity got for target for block 4:  0.4
Probablity got for foil for block 4:  0.04
Accuracy got for free recall for block 4:  0.96
Probablity got for target for block 5:  0.35
Probablity got for foil for block 5:  0.045454545454545456
Accuracy got for free recall for block 5:  0.9545454545454546
Probablity got for target for block 5:  0.36666666666666664
Probablity got for foil for block 5:  0.043478260869565216
Accuracy got for free recall for block 5:  0.9565217391304348
Probablity got for target for block 6:  0.38333333333333336
Probablity got for foil for block 6:  0.041666666666666664
Accuracy got for free recall for block 6:  0.9583333333333334
Probablity got for target for block 6:  0.38333333333333336
Probablity got for foil for block 6:  0.041666666666666664
Accuracy got for free recall for block 6:  0.9583333333333334
Probablity got for target for block 7:  0.38333333333333336
Probablity got for foil for block 7:  0.041666666666666664
Accuracy got for free recall for block 7:  0.9583333333333334
Probablity got for target for block 7:  0.4
Probablity got for foil for block 7:  0.0
Accuracy got for free recall for block 7:  1.0
Probablity got for target for block 8:  0.38333333333333336
Probablity got for foil for block 8:  0.041666666666666664
Accuracy got for free recall for block 8:  0.9583333333333334
Probablity got for target for block 8:  0.4
Probablity got for foil for block 8:  0.04
Accuracy got for free recall for block 8:  0.96
Probablity got for target for block 9:  0.38333333333333336
Probablity got for foil for block 9:  0.041666666666666664
Accuracy got for free recall for block 9:  0.9583333333333334
Probablity got for target for block 9:  0.4
Probablity got for foil for block 9:  0.04
Accuracy got for free recall for block 9:  0.96
Probablity got for target for block 10:  0.35
Probablity got for foil for block 10:  0.045454545454545456
Accuracy got for free recall for block 10:  0.9545454545454546
Probablity got for target for block 10:  0.06666666666666667
Probablity got for foil for block 10:  0.0
Accuracy got for free recall for block 10:  1.0
Probablity got for target for block 11:  0.3333333333333333
Probablity got for foil for block 11:  0.16666666666666666
Accuracy got for free recall for block 11:  0.8333333333333334
Probablity got for target for block 11:  0.36666666666666664
Probablity got for foil for block 11:  0.08333333333333333
Accuracy got for free recall for block 11:  0.9166666666666666
Probablity got for target for block 12:  0.38333333333333336
Probablity got for foil for block 12:  0.041666666666666664
Accuracy got for free recall for block 12:  0.9583333333333334
```

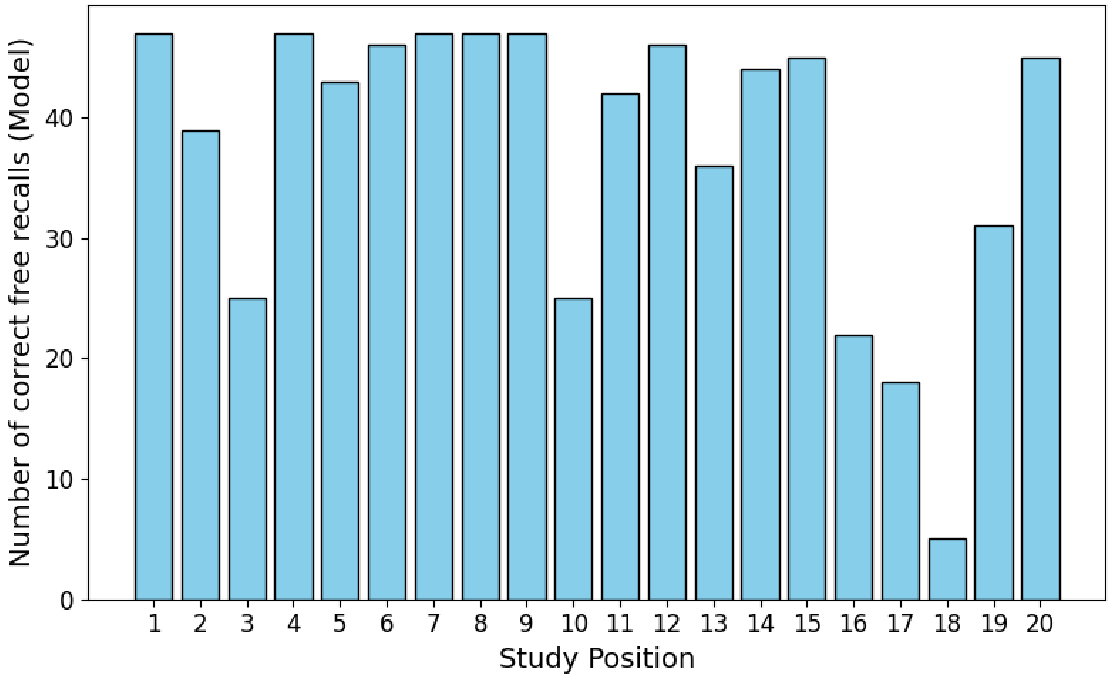
```
Probablity got for target for block 12:  0.38333333333333336
Probablity got for foil for block 12:  0.041666666666666664
Accuracy got for free recall for block 12:  0.9583333333333334
Probablity got for target for block 13:  0.35
Probablity got for foil for block 13:  0.045454545454545456
Accuracy got for free recall for block 13:  0.9545454545454546
Probablity got for target for block 13:  0.25
Probablity got for foil for block 13:  0.375
Accuracy got for free recall for block 13:  0.625
Probablity got for target for block 14:  0.36666666666666664
Probablity got for foil for block 14:  0.08333333333333333
Accuracy got for free recall for block 14:  0.9166666666666666
Probablity got for target for block 14:  0.36666666666666664
Probablity got for foil for block 14:  0.08333333333333333
Accuracy got for free recall for block 14:  0.9166666666666666
Probablity got for target for block 15:  0.38333333333333336
Probablity got for foil for block 15:  0.041666666666666664
Accuracy got for free recall for block 15:  0.9583333333333334
Probablity got for target for block 15:  0.36666666666666664
Probablity got for foil for block 15:  0.043478260869565216
Accuracy got for free recall for block 15:  0.9565217391304348
Probablity got for target for block 16:  0.36666666666666664
Probablity got for foil for block 16:  0.043478260869565216
Accuracy got for free recall for block 16:  0.9565217391304348
Probablity got for target for block 16:  0.0
Probablity got for foil for block 16:  1.0
Accuracy got for free recall for block 16:  0.0
Probablity got for target for block 17:  0.11666666666666667
Probablity got for foil for block 17:  0.3
Accuracy got for free recall for block 17:  0.7
Probablity got for target for block 17:  0.18333333333333332
Probablity got for foil for block 17:  0.35294117647058826
Accuracy got for free recall for block 17:  0.6470588235294118
Probablity got for target for block 18:  0.0
Probablity got for foil for block 18:  1.0
Accuracy got for free recall for block 18:  0.0
Probablity got for target for block 18:  0.08333333333333333
Probablity got for foil for block 18:  0.0
Accuracy got for free recall for block 18:  1.0
Probablity got for target for block 19:  0.35
Probablity got for foil for block 19:  0.045454545454545456
Accuracy got for free recall for block 19:  0.9545454545454546
Probablity got for target for block 19:  0.16666666666666666
Probablity got for foil for block 19:  0.4117647058823529
Accuracy got for free recall for block 19:  0.5882352941176471
Probablity got for target for block 20:  0.36666666666666664
Probablity got for foil for block 20:  0.043478260869565216
Accuracy got for free recall for block 20:  0.9565217391304348
```

```
Probablity got for target for block 20:  0.38333333333333336
Probablity got for foil for block 20:  0.041666666666666664
Accuracy got for free recall for block 20:  0.9583333333333334
```

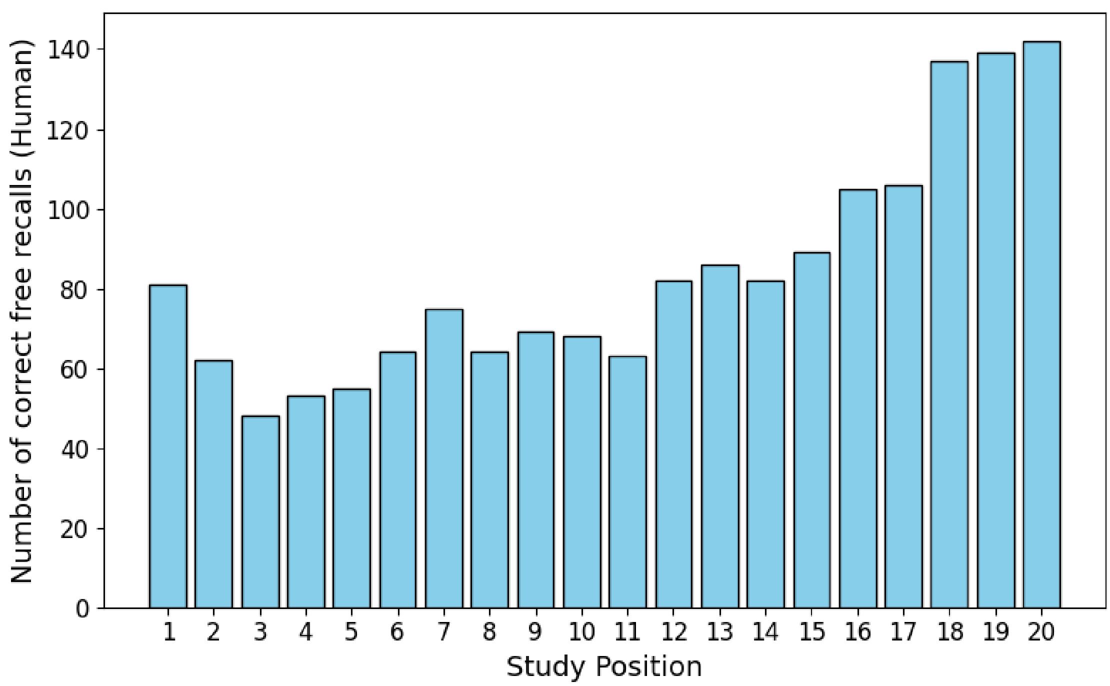
[47]:

	study.pos.left	num_corr
0	1	47
1	2	39
2	3	25
3	4	47
4	5	43
5	6	46
6	7	47
7	8	47
8	9	47
9	10	25
10	11	42
11	12	46
12	13	36
13	14	44
14	15	45
15	16	22
16	17	18
17	18	5
18	19	31
19	20	45

[48]:
```
# Plot the graph
create_plot(x=model_acc_by_pos['study.pos.left'],␣
 ↪y=model_acc_by_pos['num_corr'], task_name='free recalls (Model)',␣
 ↪task_type='Number of')
```



[49]:
```
create_plot(x= free_acc_by_pos['study.pos.left'], y = free_acc_by_pos['prob'],␣
↪task_name= 'free recalls (Human)', task_type = 'Number of')
```



Comparison of Results from Free Recall Task

Model Performance (Graph 1)

- The model displays **inconsistent recall** across study positions.
- There is **no clear trend** of better recall at the start or end of the list.

Human Performance (Graph 2)

- Humans show a **primacy effect** (better recall of words from the beginning of the list) and a **recency effect** (better recall of words from the end of the list).
- Recall steadily increases toward the end, with **study positions 18-20** showing the highest number of correct recalls.

Inference

Key Differences

- The **model lacks primacy and recency effects** seen in human recall, resulting in inconsistent performance.

- Humans use natural memory strategies:
  - **Primacy effect:** Rehearsing early items more frequently.
  - **Recency effect:** Recalling recent items still in working memory.

-

  ## 0.3 The model appears to treat all study positions equally, missing these natural biases.

Conclusion

- Humans **outperform the model** due to structured recall patterns driven by cognitive processes (primacy and recency effects).

0.3.1 5. Lexical Decision Prompt

[50]:

```python
def generate_pseudo_words(real_words, num_pseudo_words=30):
    """
    Generates a list of pseudo-words based on the given real words.

    Args:
        real_words: List of real words.
        num_pseudo_words: Number of pseudo-words to generate.

    Returns:
        A list of pseudo-words.
    """
```

```python
    vowels = "aeiou"
    consonants = "bcdfghjklmnpqrstvwxyz"
    pseudo_words = []

    while len(pseudo_words) < num_pseudo_words:
      word_length = random.randint(5, 8)
      pseudo_word = ""

      for _ in range(word_length):
        if random.random() < 0.1:
          pseudo_word += random.choice(vowels)
        else:
          pseudo_word += random.choice(consonants)

      # Check if the generated word is unique and not a real word
      if pseudo_word not in real_words and pseudo_word not in pseudo_words:
        pseudo_words.append(pseudo_word.upper())

    return pseudo_words
```

```python
[51]: def llm_lexical_decision_test(block_id, stimuli):
        """
        Simulates the lexical decision task.

        Args:
            block_id (int): The block identifier.
            stimuli (list): A list of words and pseudo-words for testing.

        Returns:
            list: A list of tuples containing the stimulus and the model's response␣
      ↪(WORD/NOT A WORD).
        """
        prompts = []
        for stimulus in stimuli:
            prompt = (
                f"You are tasked with deciding if the following is a real word or a␣
      ↪pseudo-word. "
                f"Respond with 'yes' for a real word and 'no' for a pseudo-word. "
                f"The item is: {stimulus}. Respond as QUICKLY as possible."
            )
            prompts.append(prompt)

        # Generate LLM responses for each stimulus
        results = []
        for stimulus, prompt in zip(stimuli, prompts):
            input_ids = tokenizer.encode(prompt, return_tensors="pt").to(device)
            output = model.generate(input_ids, max_length=50)
```

```
        response = tokenizer.decode(output[0], skip_special_tokens=True).strip()

        # Append the result as a tuple (stimulus, response)
        results.append((stimulus, response))

    # Log the task to context
    context[block_id]["tasks"].append("Lexical Decision")

    return results
```

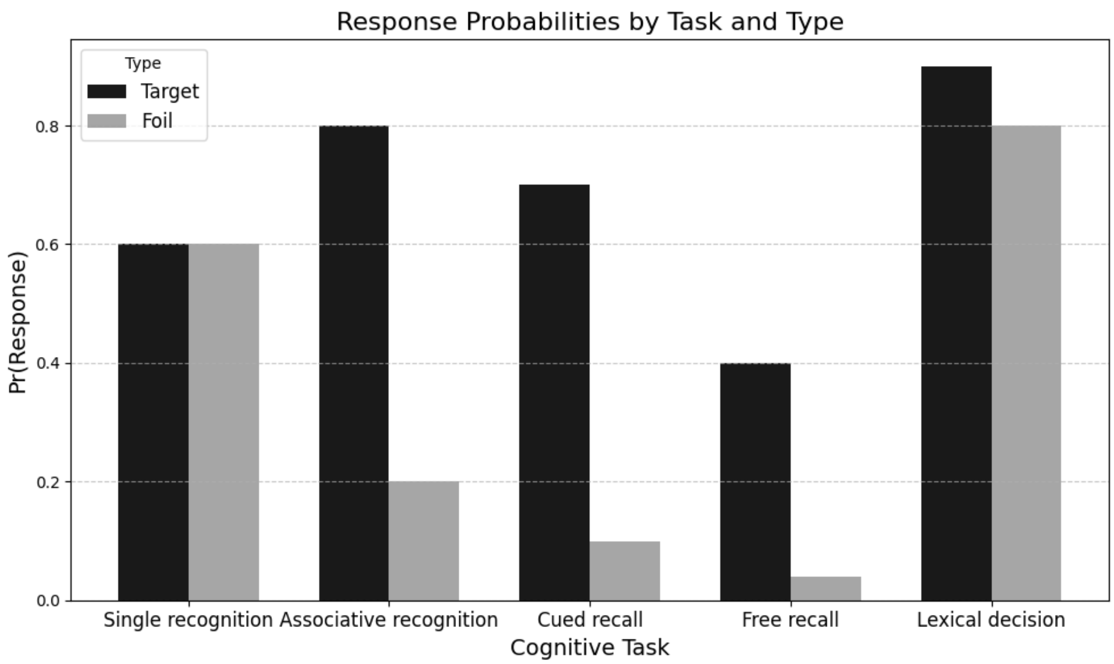
[52]:
```
def testing_lexical(block_id, study_list):
  separated_list = get_separated_words(random.sample(study_list, 10))
  target_list = random.sample(separated_list, 10)
  pseudo_words = generate_pseudo_words(separated_list)
  pseudo_list = random.sample(pseudo_words, 10)

  stimuli = target_list + pseudo_list
  random.shuffle(stimuli)
  results = llm_lexical_decision_test(block_id= block_id, stimuli=stimuli)
  correct_decisions = 0
  correct_foils = 0

  for word, response in results:
    if word in target_list and response.upper() == "YES":
      correct_decisions += 1
    elif word in pseudo_list and response.upper() == "NO":
      correct_foils += 1

  p_target = correct_decisions / len(target_list)
  p_foil = correct_foils / len(pseudo_list)
  accuracy = (correct_decisions + correct_foils) / len(stimuli)

  print(f"Probability for Targets (p_target): {p_target}")
  print(f"Probability for Foils (p_foil): {p_foil}")
  print(f"Accuracy for lexical decision: {accuracy}")

  return accuracy, p_target, p_foil
```

[53]:
```
accuracy_lexical, p_target_lexical, p_foil_lexical = testing_lexical(1,␣
 ↪study_data_1)
target_means.append(p_target_lexical)
foil_means.append(p_foil_lexical)
```

```
Probability for Targets (p_target): 0.9
Probability for Foils (p_foil): 0.8
Accuracy for lexical decision: 0.85
```

0.4 Graph for Figure 3

[66]:
```python
import matplotlib.pyplot as plt
import numpy as np

# Data for plotting
tasks = ['Single recognition', 'Associative recognition', 'Cued recall', 'Free␣
 ↪recall', 'Lexical decision']

# Bar width
bar_width = 0.35

# X-axis positions
x = np.arange(len(tasks))

# Create the plot
fig, ax = plt.subplots(figsize=(10, 6))

# Bars for "Target"
rounded_target_means = [round(value, 2) for value in target_means]
ax.bar(x - bar_width/2, rounded_target_means, bar_width, label='Target',␣
 ↪color='black', alpha=0.9, capsize=5)

# Bars for "Foil"
rounded_foil_means = [round(value, 2) for value in foil_means]
ax.bar(x + bar_width/2, rounded_foil_means, bar_width,label='Foil',␣
 ↪color='gray', alpha=0.7, capsize=5)

# Add labels, title, and legend
ax.set_ylabel('Pr(Response)', fontsize=14)
ax.set_xlabel('Cognitive Task', fontsize=14)
ax.set_title('Response Probabilities by Task and Type', fontsize=16)
ax.set_xticks(x)
ax.set_xticklabels(tasks, fontsize=12)
ax.legend(title='Type', fontsize=12)

# Customize grid and layout
ax.yaxis.grid(True, linestyle='--', alpha=0.7)
plt.tight_layout()

# Show the plot
plt.show()
```



Observations-

Cognitive Tasks and Response Types

- **Target** responses consistently show higher probabilities compared to **Foil** responses across all cognitive tasks.
- The gap between **Target** and **Foil** responses varies by task.

Task-Specific Observations

1. **Single Recognition Task**
    - Probability of **Foil** and **Target** responses are almost similar.
2. **Associative Recognition Task**
    - **Target** responses dominate with a probability close to 1.0, while **Foil** responses are much lower.
3. **Cued Recall Task**
    - **Target** responses remain high (close to 0,8), while **Foil** responses are moderately low.
4. **Free Recall Task**
    - **Target** responses are moderate, with very low probabilities of **Foil** responses, suggesting a lower error rate in free recall.
5. **Lexical Decision Task**
    - **Target** responses are a bit higher than **Foil** responses.

Summary

- The model's performance is task-dependent and shows varying sensitivity to **Target** and **Foil** stimuli.

0.5 Overall Observations

For the 5 memory congitive test, we have got the accuracy for human and Model, we have not considered response time in both cases since it can not be calculated for llm, considering gpu and cpu model device changes.

The model which we have selected is google/flan-t5-large

Source: https://huggingface.co/google/flan-t5-large

From the paper of this model:

The primary use is research on language models, including: research on zero-shot NLP tasks and in-context few-shot learning NLP tasks, such as reasoning, and question answering; advancing fairness and safety research, and understanding limitations of current large language models

All tasks are performed based on the description provided in the paper.

The comparison of model and human performance across various cognitive tasks reveals notable differences in accuracy. Here's a summary of the findings:

0.5.1 1. Single Item Recognition:

- **Human Accuracy:** 83.51%
- **Model Accuracy:** 60%
- **Observation:** Humans significantly outperform the model in recognizing individual items.

0.5.2 2. Associative Recognition:

- **Human Accuracy:** 80.34%
- **Model Accuracy:** 65%
- **Observation:** Humans perform better in associative recognitions.

0.5.3 3. Cued Recall:

- **Human Accuracy:** 31.6%
- **Model Accuracy:** 70%
- **Observation:** The model excels in cued recall, achieving a much higher accuracy than humans.

0.5.4 4. Free Recall:

- **Human Accuracy:** 7.79%
- **Model Accuracy:** 96%
- **Observation:** The model outperforms humans in free recall tasks by a significant margin.

0.5.5 5. Lexical Decision:

- **Human Accuracy:** 95.78%
- **Model Accuracy:** 85%
- **Observation:** Both humans and the model perform highly in lexical decision tasks, with humans leading by a slight margin.

0.6 Conclusion:

- **Strengths:** The model excels in tasks such as Cued Recall and Free Recall, showing a strong performance where human recall abilities are lower.
- **Challenges:** The model faces challenges in tasks that require associative recognition and single-item recognition, where humans perform better.
- **Overall Performance:** While the model performs impressively in certain tasks (like Cued Recall and Free Recall), it generally lags behind humans in more complex associative recognition and single-item recognition tasks. The differences highlight the specific strengths and weaknesses of both human and machine processing in memory-related tasks.