Procedural Learning with Graded Entropy

Cherrie Chang[1] and **Tianyi Li**[2] and **Joshua Hartshorne**[1]
[1]Department of Communication Sciences and Disorders, Mass General Hospital Institute of Health Professions, Charlestown, MA, USA
[2]Department of Psychology and Neuroscience, Boston College, Chestnut Hill, MA, USA
cherriechang@gmail.com, lidzr@bc.edu, JKHartshorne@gmail.com

```
#| label: Load R libraries
#| echo: false
#| message: false

library(here)
library(osfr)
library(dplyr)
library(tidyr)
library(tidyverse)
library(zoo)
library(jsonlite)
library(clipr)
library(reticulate)
library(lmerTest) # show p value in lmer
library(blme)
library(purrr)

np <- import("numpy")
```

Abstract

Your abstract text here. The abstract should be one paragraph. Following the abstract should be keywords.

Keywords: procedural learning, implicit learning, statistical learning

Introductino

How many times have you typed “form” instead of “from”? Or ended a word with “-tino” instead of “-tion”, perhaps even in an important, embarrassingly glaring context? The typos we make are not random; they reflect the statistical patterns in the language we use. Experienced typists learn that some keystroke sequences are highly probable, causing their fingers to automatically perform ones like “for” mistakenly, or perform them so quickly they slip up, like in the case of “-tino”. Yet typists cannot consciously report these probabilities or articulate their finger movements at the speed of execution - this knowledge is expressed in the automatic motor performance of typing, not in conscious awareness. This implicit form of learning, where statistical regularities are unconsciously acquired and expressed as behavior rather than declared knowledge, is called procedural learning (Squire (2004), Nissen & Bullemer (1987)).

Procedural learning is not limited to typing. It underlies skill acquisition across many domains, from proprioceptive skills like riding a bike to perceptual-cognitive skills like reading mirrored text (Kassubek et al. (2001)). More recently, procedural learning has been proposed to support increasingly diverse and sophisticated abilities, notably grammar acquisition in language (Ullman & Pierpont (2005), Lammertink et al. (2017), Lum et al. (2014), Kidd & Arciuli (2016)). These proposals invite closer scrutiny of its scalability: can the procedural learning system handle the large-scale, complex statistical structures that characterize domains like grammar? Consider a suggestive historical case: while QWERTY keyboards (26 letter keys) enabled fluent touch typing through procedural learning, Chinese typewriters (trays of 2,000+ characters) never achieved comparable automaticity and were eventually abandoned (for QWERTY keyboards, among others!) (Mullaney (2017)). This raises a fundamental question: does procedural learning fail beyond a certain level of statistical complexity and state space size? As a first step toward answering these questions, we examine whether procedural learning tracks fine-grained statistical patterns - both the predictability of individual transitions and the uncertainty of different states - and how it scales across state space sizes from 4 to 8 positions.

The primary method for measuring procedural learning has been the Serial Reaction Time Task (SRTT; Nissen & Bullemer (1987)). In the SRTT, participants respond to stimuli appearing in one of four positions on screen using key presses. Unbeknownst to them, the sequence of positions follows a deterministic pattern. Over time, reaction times (RT) decrease for patterned trials relative to random trials, demonstrating procedural learning of the key press sequence corresponding to the positional sequence’s deterministic pattern. While the SRTT provided early evidence for procedural learning, it suffers from a critical confound: participants often develop explicit awareness of the sequence, making it difficult to isolate implicit procedural learning effects (Song et al. (2008), Lustig (2022)). This has contributed to low test-retest reliability of the task (West et al. (2018), Oliveira et al. (2023)).

The Alternating Serial Reaction Time Task (ASRT; Howard Jr. & Howard (1997)) addressed these issues by introducing a probabilistic design. In ASRT, deterministic (d) and random (r) trials alternate (e.g., d-r-d-r), resulting in positional sequences with two levels of frequency (high vs. low). This probabilistic design minimizes explicit awareness and dramatically improves reliability (Farkas et al. (2024), Oliveira et al. (2023)). However, a key limitation of ASRT is that its bimodal frequency distribution represents an overtly simplistic statistical structure, lacking the graded frequency distributions found in real-world domains like language. Finally, across both SRTT and ASRT, experiment designs have mostly tested sequences with four possible positions (e.g., Janacsek et al. (2012), Hedenius et al. (2011)), despite real-world domains where procedural learning is implicated often involving far larger state spaces. Whether procedural learning can track graded statistical structures, and whether it scales to larger state spaces, remains untested.

Here, we introduce a new paradigm that retains ASRT’s probabilistic design but introduces a graded statistical structure with a smooth continuum of transition probabilities. We construct transition matrices to generate positional sequences where each position has a different entropy, ranging from deterministic (low entropy) to uniformly distributed (high entropy). We also vary the number of positions (4, 5, 6, 7, 8) to test whether procedural learning scales with state space size. This allows us to ask: (1) Are learners sensitive to fine-grained differences in transition probabilities (surprisal) and state-level uncertainty (entropy), beyond simple high vs. low frequency contrasts? (2) Do learning trajectories change as the number of positions increases? Across 219 participants, we find evidence that procedural learning is sensitive to graded statistical structure, and that this sensitivity emerges early and strengthens over time. These results suggest that procedural learning is more flexible than prior paradigms can reveal, with implications for understanding how procedural learning operates in naturalistic environments.

Hypotheses

We make the following predictions: First, if participants are learning the statistical structure and not merely practicing motor responses, trial-level RT and accuracy will be predicted by surprisal—the negative log probability of the observed transition—such that higher-surprisal (less predictable) transitions elicit slower and less accurate responses (H1). Second, previous positional entropy—the entropy of the transition distribution from the previous position—will modulate performance independently, with higher-entropy (more uncertain) previous positions leading to slower and less accurate responses (H2). We further predict that surprisal effects will be stronger following low-entropy positions, where predictions are more confident and prediction errors more costly (surprisal × entropy interaction). Third, these effects will be attenuated as the number of positions increases, reflecting greater difficulty in learning more complex statistical structures (H3). Note that number of positions is confounded with entropy range in our design, so H3 effects may partially reflect differences in entropy distributions across conditions.

Method

Participants

```
#| label: Demographic data
#| echo: false

df.demographics <- list.files(here("demographic-dat
  lapply(function(f) {
    matrix_size <- gsub('.*_(\\dx\\d)\\.csv', '\\1
    read.csv(f) %>%
      mutate(Matrix.size = matrix_size)
  }) %>%
  bind_rows() %>%
  filter(Status=="APPROVED")

overall_N <- nrow(df.demographics)
overall_mean_age <- mean(as.numeric(df.demographics
overall_median_age <- median(as.numeric(df.demograp
overall_sd_age <- sd(as.numeric(df.demographics$Age
overall_age_range <- range(as.numeric(df.demographi

N_nationalities <- length(unique(df.demographics$Na
N_languages <- length(unique(df.demographics$Langua
```

```
# Gender distribution
gender_counts <- table(df.demographics$Sex)
n_female <- gender_counts["Female"]
n_male <- gender_counts["Male"]
n_other <- sum(gender_counts[!names(gender_counts) %in% c('

df.demographics.group_summary <- df.demographics %>%
  group_by(Matrix.size) %>%
  summarise(N = n(),
            mean_age = mean(as.numeric(Age), na.rm=TRUE),
            sd_age = sd(as.numeric(Age), na.rm=TRUE),
            min_age = min(as.numeric(Age), na.rm=TRUE),
            max_age = max(as.numeric(Age), na.rm=TRUE))

group_N <- first(df.demographics.group_summary$N)
N_5x5 <- df.demographics.group_summary %>%
  filter(Matrix.size=="5x5") %>%
  pull(N)
```

We recruited `r overall_N` participants (`r n_female` female, `r n_male` male) to complete a web-based experiment using the online crowd-sourcing platform Prolific, screening for participants with an approval rating above 95% from previous studies and limiting participation to be on desktop or laptop computers. Participants ranged in age from `r overall_age_range[1]` to `r overall_age_range[2]` years (μ=`r round(overall_mean_age, 1)`, σ=`r round(overall_sd_age, 1)`), representing `r N_nationalities` nationalities and `r N_languages` primary languages. Participants were compensated at $12.00 per hour, with median completion time varying by condition (~20-40 minutes). All participants provided informed consent prior to the experiment. The study was approved by the Institutional Review Board at the MGH Institute of Health Professions.

Experiment Design

```
#| label: Experiment configs that stay constant across part
#| echo: false
#| include: false

EXPERIMENT_CONFIG <- list(
  osf_id_4x4 = "Emfhw",
  osf_id_5x5 = "Fraxt",
  osf_id_6x6 = "jx48y",
  osf_id_7x7 = "92jq7",
  osf_id_8x8 = "Tx3h7",
  key_mapping_4pos = c("d","f","j","k"),
  key_mapping_5pos = c("s","d","f","j","k"),
  key_mapping_6pos = c("s","d","f","j","k","l"),
  key_mapping_7pos = c("a","s","d","f","j","k","l"),
  key_mapping_8pos = c("a","s","d","f","j","k","l",";"),
  n_blocks = 20,
  rsi = 120,
  error_feedback_duration = 200,
  error_tone_duration = 100,
  correct_feedback_duration = 200,
  estimated_trial_duration = 500,
  accuracy_threshold = 0.65,
```

Table 1: Key mappings by number of positions

n positions	keys (left to right)
4	{D, F, J, K}
5	{S, D, F, J, K}
6	{S, D, F, J, K, L}
7	{A, S, D, F, J, K, L}
8	{A, S, D, F, J, K, L, ;}

```
    rt_threshold = 1000
)
```

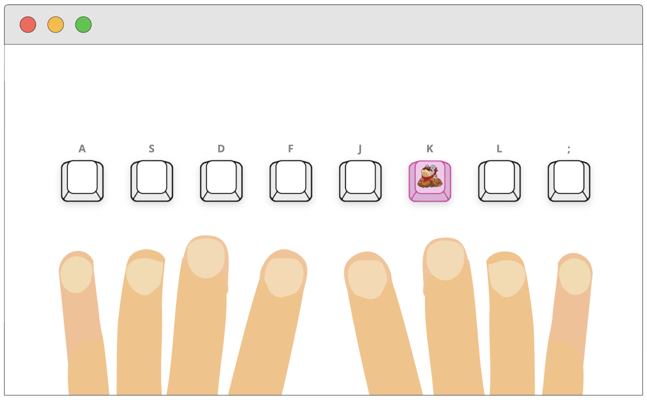

Figure 1: An example trial screen in the 8-position condition of the experiment

Like other serial reaction time task designs, our experiment tasks participants to respond to a visual stimulus that appears in one of several evenly-spaced positions on screen as quickly and accurately as possible by pressing a corresponding key on the keyboard. In our implementation, this visual stimulus is a mole garbed in a bright red bib and matching sunglasses (Figure 1). Participants were evenly divided into 5 groups of `r group_N` per condition, with the conditions differing in the number of positions (4, 5, 6, 7, 8) the mole can appear in. (**tab:key-mapping?**) shows the position-to-key mapping for each condition.

A participant is first given a text-based tutorial accompanied by an animated demonstration on which key to press in response to each position, then instructed to work through a set of practice trials where each position is visited twice in random order. After the practice section, the participant moves on to complete `r EXPERIMENT_CONFIG$n_blocks` blocks of trials, each separated with a self-paced break. Each block consists of 10 trials per position. This design choice results in longer blocks for conditions with more positions, but ensures each position is visited an equal number of times on average across conditions. In both practice and main trials, participants are given feedback on correctness via a pop-up short message (a checkmark vs. "Try again!" + an error tone), and allowed to retry each failed trial until they respond correctly. Following standard SRTT protocol, there is also a 120ms response-stimulus interval (RSI) between trials (Nissen & Bullemer (1987), Janacsek et al. (2012), Howard et al. (2004)). We find during piloting that allowing retries, combined with the `r EXPERIMENT_CONFIG$rsi`ms RSI, prevents participants from making compensatory errors, where they unintentionally press the wrong key in the next trial in an attempt to correct their current trial. We record response time, keyboard response and correctness for each trial. After each block, participants are given adaptive feedback based on their accuracy and speed. At the end of the experiment, participants complete a brief questionnaire probing their explicit awareness of any patterns in the mole's appearances (Table).

In addition to our dapper mole, a number of design choices were made to facilitate participant understanding and engagement. During the practice phase, each position on screen is labelled with its corresponding key character (e.g. "A", "S", "D") to help familiarize participants with the keyboard mappings. These key labels disappear in the main trials to prevent explicitly encoding of the sequence of positions via their character labels, but reappear during retry of failed trials. To make the correspondence between the positions on screen and keyboard keys as automatic and intuitive as possible, we chose to use conventional QWERTY-based finger layouts for the key mappings across conditions ((**tab:key-mapping?**)). We found during piloting that while this decision helped reduce cognitive load from learning novel finger placements, the larger gap between keys "F" and "J" on the keyboard introduced a discrepancy between the evenly-spaced on-screen positions and the unevenly spaced keyboard keys. This made participants more prone to making errors in the positions towards the middle of the screen compared to those on the outer edges, especially when number of positions increases [TODO: maybe run stats on pilot to find this effect]; and was not mitigated by shifting either hand's placement to close the extra gap, because the positions were still corresponding half to one hand and half to the other. After further piloting testing different visual aids, we settled on adding two hands visually to the bottom of the screen, each finger aligned with its target position (Figure). We received verbal feedback that this visual aid helped participants better map the spatial layout of the on-screen positions to their corresponding keys. Lastly, the visual design of the "positions" on screen mimicked blank 3D keyboard keys that light up in pink when the mole appears on top, and look visibly "pressed down" when the participant presses the corresponding keyboard key. This design choice links the on-screen positions diegetically to the physical keyboard keys to maximally reduce mental distance between the two. This is important to enforce as much learning via implicit motor memory as possible, rather than via visual-spatial memory of the on-screen positions.

Transition Matrices

```
#| label: Original transition matrices
#| echo: false
#| include: false

# Shannon entropy (bits) of a single probability vector (s
row_entropy <- function(prob_vec) {
  p <- prob_vec[prob_vec > 0]
  -sum(p * log2(p))
```

```
}

# Calculate positional entropy for each row in a transition matrix
positional_entropy <- function(mat) {
  apply(mat, MARGIN=1, row_entropy)
}

transition_matrix_4x4 <- matrix(unlist(np$load(here("assets/transition-matrices/matrix_4x4.npy"))), nrow=4, by
transition_matrix_5x5 <- matrix(unlist(np$load(here("assets/transition-matrices/matrix_5x5.npy"))), nrow=5, by
transition_matrix_6x6 <- matrix(unlist(np$load(here("assets/transition-matrices/matrix_6x6.npy"))), nrow=6, by
transition_matrix_7x7 <- matrix(unlist(np$load(here("assets/transition-matrices/matrix_7x7.npy"))), nrow=7, by
transition_matrix_8x8 <- matrix(unlist(np$load(here("assets/transition-matrices/matrix_8x8.npy"))), nrow=8, by
all_original_matrices <- list(
  "4x4" = transition_matrix_4x4,
  "5x5" = transition_matrix_5x5,
  "6x6" = transition_matrix_6x6,
  "7x7" = transition_matrix_7x7,
  "8x8" = transition_matrix_8x8
)

all_positional_entropies <- lapply(all_original_matrices, positional_entropy)
all_positional_entropies_df <- lapply(names(all_positional_entropies), function(size) {
  data.frame(
    Matrix.size = size,
    Position = 1:length(all_positional_entropies[[size]]),
    Entropy = all_positional_entropies[[size]]
  )
}) %>%
  bind_rows()

pos_1_entropy <- all_positional_entropies_df %>%
  group_by(Matrix.size) %>%
  filter(Position==1) %>%
  ungroup() %>%
  summarize(range=range(Entropy), mean=mean(Entropy), sd=sd(Entropy))

pos_n_entropy <- all_positional_entropies_df %>%
  group_by(Matrix.size) %>%
  filter(Position==max(Position)) %>%
  ungroup() %>%
  summarize(range=range(Entropy), mean=mean(Entropy), sd=sd(Entropy))

entropy_gradient_summary <- all_positional_entropies_df %>%
  group_by(Matrix.size) %>%
  summarize(entropy_diffs = list(diff(Entropy))) %>%
  ungroup() %>%
  summarize(
    range_diff = list(range(unlist(entropy_diffs))),
    mean_diff = mean(unlist(entropy_diffs)),
    sd_diff = sd(unlist(entropy_diffs))
  )
```

Our experiment deviates from other serial reaction time tasks in how the sequence of positions and its probabilistic structure is constructed. In each run of the experiment, we use a first-order Markov process to generate the sequence of positions the mole appears in. This process is defined by a transition matrix, where each entry in the matrix defines the probability of transitioning from one position to another. We designed five different transition matrices of sizes 4x4, 5x5, 6x6, 7x7, and 8x8 respectively corresponding to the five conditions in our experiment. These "original" matrices were constructed to have graded and increasing levels of entropy across rows, such that position 1 has lowest entropy-having the most predictable next-position-transitions, followed by the second, with the last position, position n, having highest entropy. For example, in the 4x4 matrix, position 1 has a high probability (~`r round(transition_matrix_4x4[1,2], 2)`) of transitioning to position 2, a low probability (~`r round(transition_matrix_4x4[1,1], 2)`) of transitioning to position 1, and never transitions to position 3 or 4; while position 4 has a more uniform distribution of transition probabilities to all other positions (to 1: ~(`r round(transition_matrix_4x4[4,1], 2)`), to 2: ~(`r round(transition_matrix_4x4[4,2], 2)`), to 3: ~(`r round(transition_matrix_4x4[4,3], 2)`), to 4: ~(`r round(transition_matrix_4x4[4,4], 2)`)). This gives rise to a gradient of entropy values across the positions, which results in a richer, more varied distributed statistical structure underlying the sequence of positions the mole appears in, allowing us to investigate how the procedural learning system differentially picks up varying levels of predictability in a given statistical structure.

Several design criteria/constraints were imposed on the construction of these original transition matrices. First, these matrices had to be doubly stochastic to ensure uniform visitation to each position over time, preventing confounding effects from position frequency on learning. Second, self-transitions (e.g. position 1 to position 1) should have near-zero probabilities to reduce data loss, as data from repeated trials have been historically thrown out from analysis in serial reaction time tasks to prevent confounding motor effects from repeating the same keystroke [CITE]. Third, the graded entropy levels across the positions should translate to graded probabilities across bigrams as well, as having more fine-grained, distributed levels of transition probabilities is our experiment's key distinction from other SRTT tasks, such as the ASRT paradigm which only has 2 levels of trigram probability. Fourth, the increase in entropy across positions (i.e. rows) should be significant between one another and as linear as possible to ensure smooth a smooth gradient with meaningfully distinct levels in predictability across positions.

Strictly satisfying all these constraints at once turned out to be not only non-trivial, but mathematically impossible for our range of n_positions (i.e. matrix sizes), so we wrote an optimization algorithm that explores the candidate space for each matrix size, treating these constraints as soft penalties to find a global minimum of the total constraint violation, balancing trade-offs among the competing constraints.

The five original matrices we constructed using this algorithm had these properties: 1) near fully doubly stochastic—all row and column sums sum to 1 (±0.0001); 2) approach a uniform stationary distribution after 7 timesteps where the stationary visitation for each state (position) are within 0.001 of each other; 3) had a graded entropy structure across states—each matrix had a positional entropy range of `r round(pos_1_entropy$range[1], 3)`-`r round(pos_1_entropy$range[2], 3)` bits for position 1 (μ=`r round(pos_1_entropy$mean[1], 3)`, σ=`r round(pos_1_entropy$sd[1], 3)`), and r round(pos_n_entropy$range[1], 3)-r round(pos_n_entropy$range[2], 3) bits for position n (μ=r round(pos_n_entropy$mean[1], 3), σ=r round(pos_n_entropy$sd[1], 3)). Within a matrix each position increases in entropy from the previous by r round(entropy_gradient_summary$range_diff[[1]][1], 3)-r round(entropy_gradient_summary$range_diff[[1]] 3) bits (μ=r round(entropy_gradient_summary$mean_diff[3), σ=r round(entropy_gradient_summary$sd_diff[1], 3)); and 4) avoided self loops as much as possible—all of the matrices had <0.08 probabilities in the left diagonal up until the last two positions. Within each matrix size condition, all participants experience the same underlying entropy gradient across positions, but the mapping between screen positions and entropy levels is randomized through a double permutation procedure that shuffles the original matrix (of that size) using the same random permutation. This preserves the statistical properties of the original matrix—transition probabilities and entropy values—but decouples entropy levels from specific screen positions. The position sequence for each participant is subsequently generated using their individual shuffled transition matrix, so the sequence each participant experiences is different both within and between matrix size conditions.

Statistical Analysis

```
#| label: Retrieve data from OSF
#| echo: false
#| include: false
#| eval: false

# NOTE: Set eval=true only when you need to downloa
# Once data is downloaded locally, keep eval=false

files_4x4 <- osf_retrieve_node(EXPERIMENT_CONFIG$os
  osf_ls_files(n_max = Inf) %>%
  osf_download(path = here("data/4x4-data"), confli

files_5x5 <- osf_retrieve_node(EXPERIMENT_CONFIG$os
  osf_ls_files(n_max = Inf) %>%
  osf_download(path = here("data/5x5-data"), confli

files_6x6 <- osf_retrieve_node(EXPERIMENT_CONFIG$os
  osf_ls_files(n_max = Inf) %>%
  osf_download(path = here("data/6x6-data"), confli

files_7x7 <- osf_retrieve_node(EXPERIMENT_CONFIG$os
  osf_ls_files(n_max = Inf) %>%
  osf_download(path = here("data/7x7-data"), confli

files_8x8 <- osf_retrieve_node(EXPERIMENT_CONFIG$os
  osf_ls_files(n_max = Inf) %>%
  osf_download(path = here("data/8x8-data"), confli
```

```
#| label: List local data files
#| echo: false
#| include: false

# But there is some bug in osfr that causes one fil
```

```
files_4x4 <- list.files("data/4x4-data")
files_5x5 <- list.files("data/5x5-data")
files_6x6 <- list.files("data/6x6-data")
files_7x7 <- list.files("data/7x7-data")
files_8x8 <- list.files("data/8x8-data")
```

```
#| label: Bind all data together
#| echo: false
#| include: false

# Cache df.raw to speed up rendering
df_raw_cache <- here("data/processed/df_raw.rds")

if (file.exists(df_raw_cache)) {
  df.raw <- readRDS(df_raw_cache)
} else {
  df.raw <- list.files(here("data/"), pattern = "*.csv", fu
    .[!grepl("pilot-data", .)] %>%
    lapply(read.csv) %>%
    bind_rows()

  # Create processed directory if it doesn't exist
  dir.create(here("data/processed"), showWarnings = FALSE,
  saveRDS(df.raw, df_raw_cache)
}
```

```
#| label: Filter out unnecessary columns
#| echo: false
#| include: false

df <- subset(df.raw, select=c(trial_index, subject_id, matr
```

```
#| label: Clean up data formatting
#| echo: false
#| include: false

# Convert NA values
df <- df %>%
  mutate(across(where(is.character), ~{
    x <- .
    x[x %in% c("NA", "na", "null", "NULL", "")] <- NA
    x
  }))

# Convert boolean values
df <- df %>%
  mutate(across(where(is.character), ~ {
    x <- tolower(.)
    if (all(x %in% c("true", "false", NA))) {
      as.logical(x)
    } else {
      .
    }
  }))

# Convert numeric values
df <- df %>%
  mutate(across(where(is.character), ~ {
    x <- .
    is_num <- suppressWarnings(!is.na(as.numeric(x)) | is.s
    if (all(is_num)) as.numeric(x) else x
  }))
```

```
#| label: Center matrix_size and block
#| echo: false
#| include: false

df <- df %>%
  mutate(
    matrix_size_c=matrix_size-((max(df$matrix_size,
    block_c=block-((max(df$block, na.rm=TRUE)+min(d
  )
```

```
#| label: Fill down overall_trial, trial_in_block, block
#| echo: false
#| include: false

df <- df %>%
    group_by(subject_id) %>%
    arrange(subject_id, row_number()) %>%
    mutate(
      overall_trial = if_else(
        experiment_trial_type %in% c("feedback", "r
        zoo::na.locf(overall_trial, na.rm = FALSE),
        overall_trial  # Keep as-is
      ),
      trial_in_block = if_else(
        experiment_trial_type %in% c("feedback", "r
        zoo::na.locf(trial_in_block, na.rm = FALSE)
        trial_in_block
      ),
      block = if_else(
        experiment_trial_type %in% c("feedback", "r
        zoo::na.locf(block, na.rm = FALSE),
        block
      )
    ) %>%
    ungroup()
```

```
#| label: Sanity check for unique subject_ids, matr                          transitio
#| echo: false
#| include: false

unique_subjects <- unique(df$subject_id)

# Per-subject constants grouped by matrix_size
df %>%
  distinct(matrix_size, subject_id, total_trials, p                          tropies) %
  group_by(matrix_size) %>%
  summarize(
    n_subjects = n_distinct(subject_id),
    total_trials = unique(total_trials),
    practice_trials = unique(practice_trials),
    transition_matrix = list(unique(transition_matrix)),
    conditional_entropies = list(unique(conditional
  )

# Unique position sequences per matrix_size
df %>%
  group_by(matrix_size, subject_id) %>%
  summarize(
    sequence = paste(position, collapse = ","),
    .groups = "drop"
  ) %>%
  group_by(matrix_size) %>%
  summarize(
    n_subjects = n_distinct(subject_id),
    n_unique_sequences = n_distinct(sequence)
  )

df <- df[, !(names(df) %in% c("transition_matrix", "conditi

# Shows only 1 unique transition matrix and conditional ent
```

```
#| label: Separate practice, main and questionnaire sectior
#| echo: false
#| include: false

df.practice <- df[df$phase=="practice", ]
df.questionnaire <- df[df$phase=="questionnaire", ]
df <- df[df$phase=="main", ]
```

```
#| label: exclude-subjects
#| echo: false
#| include: false

# Exclude subjects that
# 1. Made too many mistakes (more than 15%)
# 2. Have too long response time (look at mean, median and
# -> exclude if too slow--the subject's median_rt > 1000ms
# -> or if too noisy--more than 20% of their trials are 3*r
accuracy_excluded_subjects <- df %>%
  filter(!is.na(correct)) %>%
  group_by(subject_id) %>%
  summarize(error_rate = mean(!correct)) %>%
  filter(error_rate > 0.10) %>%
  pull(subject_id)

rt_excluded_subjects <- df %>%
  filter(
    (experiment_trial_type == "stimulus" | experiment_trial
    & !is.na(rt)) %>%
  group_by(subject_id) %>%
  summarize(
    median_rt = median(rt),
    outlier_prop = mean(rt > median(rt) + 3*mad(rt))
  ) %>%
  filter(median_rt > 1000 | outlier_prop > 0.2) %>%
  pull(subject_id)

df <- df %>%
  filter(!subject_id %in% accuracy_excluded_subjects & !sub
```

```
#| label: Filter df down to only stimulus, non-retry trials
#| echo: false
#| include: false

df <- df %>%
  filter(experiment_trial_type=="stimulus" & !is.na(rt) & r
```

```r
#| label: Remove first trial of each block
#| echo: false
#| include: false

df <- df %>%
  group_by(block, subject_id) %>%
  slice(2:n())
```

```r
#| label: Make a new column for log(rt), previous position entropy, prev_entropy_c, surprisal_c, is_repetition
#| echo: false
#| include: false

# Cache fully processed df
df_processed_cache <- here("data/processed/df_processed.rds")

if (file.exists(df_processed_cache)) {
  df <- readRDS(df_processed_cache)
} else {
  df <- df %>%
    mutate(log_rt=log(rt)) %>%
    mutate(is_repetition=(position==lag(position,1))) %>%
    group_by(subject_id, block) %>%
    mutate(previous_entropy=lag(conditional_entropy, 1)) %>%
    ungroup() %>%
    group_by(matrix_size) %>%
    mutate(prev_entropy_c=previous_entropy-mean(previous_entropy, na.rm=TRUE)) %>%
    mutate(surprisal_c=surprisal-mean(surprisal, na.rm=TRUE)) %>%
    ungroup()

  saveRDS(df, df_processed_cache)
}
```

```r
#| label: Make subset of df with only correct trials
#| echo: false
#| include: false

# Cache correct trials df
df_correct_cache <- here("data/processed/df_correct.rds")

if (file.exists(df_correct_cache)) {
  df.correct <- readRDS(df_correct_cache)
} else {
  df.correct <- df %>%
    filter(correct)

  saveRDS(df.correct, df_correct_cache)
}
```

Variables and Exclusion Criteria

The main independent variables are surprisal and previous positional entropy. Surprisal is defined as the negative log probability of the actual next-position given the current position [TODO: math], derived from the transition matrix used to generate the sequence for that participant. Previous positional entropy is defined as the Shannon entropy of the transition probabilities from the previous position, also derived from the transition matrix [TODO: math]. Both surprisal and previous positional entropy are mean-centered within each matrix size condition to facilitate interpretation of main effects and interactions in subsequent models. We also create a binary variable indicating whether the current position is a repetition of the previous position (i.e. self-transition). This variable is included as a covariate in subsequent models to control for potential motor effects from repeating the same keystroke.

The main dependent variables are response time (RT) and accuracy (proportion of correct responses) on non-practice/main trials. Incorrect trials, trials with RT >1000ms, and trials with RTs greater than 3 median absolute deviations above the participant's median RT (outliers) are excluded. On the participant level, participants with overall accuracy below 85%, median RT >1000ms, or with outliers making up more than 30% of total trials are excluded from analysis. We also discard data from the first trial of each block to account for blocks being implicitly treated as "restarts" and not following an existing probability distribution by participants. RTs are log-transformed to reduce skewness.

Modeling Approach

We use linear mixed effects regression (LMER) to model log-transformed RTs and generalized linear mixed effects regression (GLMER) with a binomial link function to model accuracy. Models are fit using the `lmerTest` package in R. We first test for the learning effect across blocks (H0), then examine the effects of surprisal and previous positional entropy (H1 & H2) in the final block, pruning effects where applicable to find the best-fitting effects structure before analyzing learning trajectories across blocks, and finally test for moderation by number of positions/matrix size(H3).

Models are initially fit with the maximal random effects structure supported by the design: random intercepts for participant and position, and random slopes for surprisal, previous positional entropy, their interaction, and position repetition by participant. Where maximal models fail to converge, we use bounded linear mixed models (`blmer`/`bglmer`) as a first remedy. If singularity persists, we iteratively simplify the random effects structure, removing random slopes one at a time and retaining the most complex structure that converges without singularity. Among converging models, we select the one with the lowest AIC.

For H1 and H2, we first establish the fixed effects structure using only the final block of trials, then extend to learning trajectories across blocks. Starting from a base model including surprisal, previous positional entropy, their interaction, and repetition as fixed effects (with the selected random effects structure), we use AIC-based model comparison to prune fixed effects. We first test whether repetition is significant by comparing the base model against one in which it is removed; if removing it does not increase AIC, we drop it from the base model. We then test whether the interaction term between surprisal and previous positional entropy is significant by comparing the current base model against one in which the interaction term is removed. If the interaction is not significant, we test each main effect individually by comparing models that omit surprisal or previous positional entropy as main effects. An effect is retained if removing it increases AIC.

Results

```r
#| label: Prepare data for H0 models
#| echo: false
#| eval: true

df.m_h0 <- df
df.m_h0.correct <- df.correct
```

```
#| label: H0 Learning effect lmer models
#| echo: false
# (filter for correct trials and group by matrix_si

m_h0_rt <-
  lmer(log_rt ~ block * is_repetition + (block * is
  data=df.m_h0.correct,
  REML=FALSE,
  control=lmerControl(optimizer="bobyqa")) # can ch

summary(m_h0_rt)

df.m_h0.correct <- df.correct %>%
  group_by(matrix_size)

m_h0_acc <-
  glmer(correct ~ block + is_repetition + (block |
  data=df.m_h0,
  family=binomial,
 control=glmerControl(optimizer="bobyqa")) # can ch

summary(m_h0_acc)
```

For H1 & H2, we analyze each block independently. First, we determine the effects structure by analyzing only the final block as follows: *(When the best-fit base model for surprisal and/or previous position entropy effects has been found, we can then examine learning trajectories by adding block as a continuous predictor)*

```
#| label: H1H2 Effect lmer models
#| echo: false
# (filter for last block only, for each matrix_size

df.m_h1h2.finalblock <- df %>%
  filter(block==EXPERIMENT_CONFIG[["n_blocks"]]-1)

data_list <- split(df.m_h1h2.finalblock, df.m_h1h2.

models <- map(data_list, ~ lmer(
  log_rt ~ surprisal_c * prev_entropy_c + is_repeti
    (surprisal_c * prev_entropy_c || subject_id) +
  data = .x,
  REML=FALSE,
  control=lmerControl(optimizer="bobyqa")))

model_summaries <- map(models, summary)
model_summaries[2]
```

```
model_structures_random_slope <- list(
  base = "(surprisal_c * prev_entropy_c || subject_
  main = "(surprisal_c + prev_entropy_c || subject_
  s_only = "(surprisal_c | subject_id) + (1 | posit
  e_only = "(prev_entropy_c | subject_id) + (1 | pc
  intercept = "(1 | subject_id) + (1 | position)"
)
```

```
all_models_random_slope <- map(
  model_structures_random_slope,
  ~ map(data_list, function(dat) {
    lmer(
      as.formula(
        paste0("log_rt ~ surprisal_c * prev_entropy_c + is_
      ),
      data = dat,
      REML = FALSE,
      control = lmerControl(optimizer = "bobyqa")
    )
  })
)

aic_table_random_slope <- map_dfr(
  names(all_models_random_slope),
  function(model_name) {
    map_dfr(
      names(all_models_random_slope[[model_name]]),
      function(cond) {
        m <- all_models_random_slope[[model_name]][[cond]]
        data.frame(
          model = model_name,
          matrix_size = cond,
          AIC = AIC(m),
          isSingular = isSingular(m)
        )
      }
    )
  }
)

aic_table_random_slope %>%
  filter(aic_table_random_slope$isSingular == FALSE) %>%
  group_by(model) %>%
  summarise(total_AIC = sum(AIC)) %>%
  arrange(total_AIC)
  #group_by(matrix_size) %>%
  #arrange(matrix_size, AIC)
```

```
model_structures_fixed_effect <- list(
  base = "surprisal_c * prev_entropy_c + is_repetition",
  no_interaction = "surprisal_c + prev_entropy_c + is_repet
  no_is_repetition = "surprisal_c * prev_entropy_c",
  only_main = "surprisal_c + prev_entropy_c",
  surprisal_is_repetition = "surprisal_c + is_repetition",
  entropy_is_repetition = "prev_entropy_c + is_repetition"
)

all_models_fixed_effect <- map(
  model_structures_fixed_effect,
  ~ map(data_list, function(dat) {
    lmer(
      as.formula(
        paste0("log_rt ~ ", .x, " + (surprisal_c * prev_ent
      ),
      data = dat,
      REML = FALSE,
      control = lmerControl(optimizer = "bobyqa")
    )
  })
)

aic_table_fixed_effect <- map_dfr(
  names(all_models_fixed_effect),
  function(model_name) {
    map_dfr(
      names(all_models_fixed_effect[[model_name]]),
      function(cond) {
        m <- all_models_fixed_effect[[model_name]][
        data.frame(
          model = model_name,
          matrix_size = cond,
          AIC = AIC(m),
          isSingular = isSingular(m)
        )
      }
    )
  }
)
```

```
aic_table_fixed_effect %>%
  filter(aic_table_fixed_effect$isSingular == FALSE
  group_by(model) %>%
  summarise(total_AIC = sum(AIC)) %>%
  arrange(total_AIC)
  #group_by(matrix_size) %>%
  #arrange(matrix_size, AIC)
```

```
theme_set(theme_minimal())

effects_df <- map_dfr(
  names(models),
  ~ broom.mixed::tidy(models[[.x]]) %>%
    filter(effect == "fixed" & term != "(Intercept)
    mutate(matrix_size = .x)
) %>%
  mutate(
    sig_label = case_when(
      p.value < 0.001 ~ "***",
      p.value < 0.01 ~ "**",
      p.value < 0.05 ~ "*",
      TRUE ~ ""
    )
  )

#effects_df$matrix_size <- as.numeric(effects_df$ma

ggplot(effects_df, aes(x = matrix_size, y = estimate, color = term, group = term)) +
  geom_line() +
  geom_point() +
  geom_errorbar(aes(ymin = estimate - std.error, ym
  geom_text(aes(label = sig_label), vjust = -1.2, s
  labs(x = "Matrix Size", y = "Effect Estimate", cc
  theme_minimal()
```

```
data.m_h1h2_all_blocks <- df %>% filter(!is.infinit

data_list_all_blocks <- split(data.m_h1h2_all_block
```

```
models_all <- map(data_list_all_blocks, ~ lmer(
  log_rt ~ surprisal_c * prev_entropy_c * block + block * i
    (surprisal_c + prev_entropy_c + is_repetition + block
  data = .x,
  REML=FALSE,
  control=lmerControl(optimizer="bobyqa")))

#iwalk(models_all, function(model, name) {
#  saveRDS(model, paste0("./models/all_block_per_condition/
#})

# to read
#model_names <- c("4", "5", "6", "7", "8")
#models_all <- map(model_names, ~ readRDS(paste0("./models/
#names(models_all) <- model_names

model_summaries_all_blocks <- map(models_all, summary)
model_summaries_all_blocks[5]
```

```
cur_condition <- 8
data_m4 <- subset(df, matrix_size == cur_condition & !is.in

data_list_block <- split(data_m4, data_m4$block)

# random effects: (prev_entropy_c | subject_id) + (1 | posi
models_block <- map2(names(data_list_block),
  data_list_block,
  function(name, dat) {
      model_path <- paste0("./models/per_block_condition_",
    dir_path <- dirname(model_path)
    if (!dir.exists(dir_path)) dir.create(dir_path, recursi
    if (file.exists(model_path)) {
      readRDS(model_path)
    } else {
      m <- lmer(
        log_rt ~ surprisal_c * prev_entropy_c + is_repeti
        (prev_entropy_c | subject_id) + (1 | position),
        data = dat,
        REML = FALSE,
        control = lmerControl(optimizer = "bobyqa")
      )
      saveRDS(m, model_path)
      m
    }
  }
)

model_summaries_block <- map(models_block, summary)
model_summaries_block[10]
```

```
model_structures_random_slope <- list(
  base = "(prev_entropy_c | subject_id) + (1 | position)",
  main = "(surprisal_c + prev_entropy_c || subject_id) + (1
  s_only = "(surprisal_c | subject_id) + (1 | position)",
  e_only = "(prev_entropy_c | subject_id) + (1 | position)"
  intercept = "(1 | subject_id) + (1 | position)"
)

all_models_random_slope <- map(
  model_structures_random_slope,
  ~ map(data_list, function(dat) {
    lmer(
      as.formula(
        paste0("log_rt ~ surprisal_c * prev_entropy
      ),
      data = dat,
      REML = FALSE,
      control = lmerControl(optimizer = "bobyqa")
    )
  })
)

aic_table_random_slope <- map_dfr(
  names(all_models_random_slope),
  function(model_name) {
    map_dfr(
      names(all_models_random_slope[[model_name]]),
      function(cond) {
        m <- all_models_random_slope[[model_name]][
        data.frame(
          model = model_name,
          matrix_size = cond,
          AIC = AIC(m),
          isSingular = isSingular(m)
        )
      }
    )
  }
)

aic_table_random_slope %>%
  filter(aic_table_random_slope$isSingular == FALSE) %>%
  group_by(model) %>%
  summarise(total_AIC = sum(AIC)) %>%
  arrange(total_AIC)
  #group_by(matrix_size) %>%
  #arrange(matrix_size, AIC)
```

```
effects_block <- map_dfr(
  names(models_block),
  ~ broom.mixed::tidy(models_block[[.x]]) %>%
    filter(effect == "fixed" & term != "(Intercept)
    mutate(block = as.numeric(.x))
) %>%
  mutate(
    sig_label = case_when(
      p.value < 0.001 ~ "***",
      p.value < 0.01 ~ "**",
      p.value < 0.05 ~ "*",
      TRUE ~ ""
    )
  )

ggplot(effects_block, aes(x = block, y = estimate,
  geom_line() +
  geom_point() +
  geom_errorbar(aes(ymin = estimate - std.error, ym
  geom_text(aes(label = sig_label), vjust = -1.2, s
  labs(x = "Block", y = "Effect Estimate", color =
  theme_minimal()
```

H3

```
#| label: Num positions effect lmer models
#| echo: true
#| include: true
# (filter for correct trials and group by matrix_size and b

m_matrix_size_rt <-
  blmer(-log_rt ~ matrix_size + is_repetition + (block | su
  data=data.m_h1h2,
  REML=FALSE,
  control=lmerControl(optimizer="nloptwrap")) # can change

# m_matrix_size_rt <-
#  lmer(-log_rt ~ matrix_size + is_repetition + (block | su
#       data=data.m_h1h2,
#       REML=FALSE,
#       control=lmerControl(optimizer="bobyqa")) # can char

m_matrix_size_acc <-
    bglmer(correct ~ matrix_size + is_repetition + (block
    data=data.m_h1h2,
    family=binomial,
    REML=FALSE,
    control=lmerControl(optimizer="nloptwrap")) # can chang

summary(m_matrix_size_rt)
summary(m_matrix_size_acc)
```

For each of the fixed effects used for H1-H2, recover the effect size estimate. Now, for each fixed effect: `effect_size ~ matrix_size*block`

Note that there are no random effects. We use `lm()` and get p-values in the normal way.

Models will be fit with ML for model comparison; final models refit with REML for inference.

```
#| label: Extract coefficients from per-block models
#| echo: true
#| include: true

# Extract coefficients from all per-block models across all
# This creates a dataframe with: matrix_size, block, and co

library(broom.mixed)

# Get all per-block model directories
condition_dirs <- list.files("models/", pattern = "per_bloc

# Extract coefficients from each condition's per-block mode
coef_df <- map_dfr(condition_dirs, function(dir) {
  condition_num <- gsub(".*per_block_condition_(\\d+)", "\\
  matrix_size <- paste0(condition_num, "x", condition_num)

  # Get all model files in this directory
  model_files <- list.files(dir, pattern = "model_block_.*\

  # Extract block number and coefficients from each model
  map_dfr(model_files, function(file) {
    block_num <- as.numeric(gsub(".*model_block_(\\d+)\\.rd
    # Load model and extract fixed effects
    m <- readRDS(file)
    coefs <- fixef(m)

    # Return as a row
    data.frame(
      matrix_size = matrix_size,
      matrix_size_num = as.numeric(condition_num),
      block = block_num,
      intercept = coefs["(Intercept)"],
      surprisal = coefs["surprisal_c"],
      prev_entropy = coefs["prev_entropy_c"],
      interaction = coefs["surprisal_c:prev_entropy
      is_repetition = coefs["is_repetitionTRUE"]
    )
  })
})

# View structure
head(coef_df)
summary(coef_df)
```

```
#| label: H3 models - matrix size moderation
#| echo: true
#| include: true

# H3: Test whether surprisal, entropy, and interact
# are moderated by matrix_size and block

# Model 1: Surprisal effect ~ matrix_size * block
m_h3_surprisal <- lm(surprisal ~ matrix_size_num *
summary(m_h3_surprisal)

# Model 2: Previous entropy effect ~ matrix_size *
m_h3_entropy <- lm(prev_entropy ~ matrix_size_num *
summary(m_h3_entropy)

# Model 3: Interaction effect ~ matrix_size * block
m_h3_interaction <- lm(interaction ~ matrix_size_nu
summary(m_h3_interaction)

# Model 4: Repetition effect ~ matrix_size * block
m_h3_repetition <- lm(is_repetition ~ matrix_size_n
summary(m_h3_repetition)
```

```
#| label: H3 visualization
#| echo: false
#| include: true

# Create plots directory if it doesn't exist
dir.create("plots", showWarnings = FALSE)

# Visualize how effects change across matrix size and block

# Plot surprisal effect
p1 <- ggplot(coef_df, aes(x = block, y = surprisal, color = matrix_size, group = matrix_size)) +
  geom_smooth(method = "lm", se = TRUE, alpha = 0.2) +
  geom_point(alpha = 0.3) +
  labs(title = "H1: Surprisal Effect Across Learning",
       x = "Block", y = "Surprisal Coefficient", color = "Matrix Size") +
  theme_minimal()
ggsave("plots/h3_surprisal_by_block.png", p1, width = 8, he
print(p1)

# Plot entropy effect
p2 <- ggplot(coef_df, aes(x = block, y = prev_entropy, colo
  geom_smooth(method = "lm", se = TRUE, alpha = 0.2) +
  geom_point(alpha = 0.3) +
  labs(title = "H2: Entropy Effect Across Learning",
       x = "Block", y = "Previous Entropy Coefficient", col
  theme_minimal()
ggsave("plots/h3_entropy_by_block.png", p2, width = 8, heig
print(p2)

# Plot interaction effect
p3 <- ggplot(coef_df, aes(x = block, y = interaction, color
  geom_smooth(method = "lm", se = TRUE, alpha = 0.2) +
  geom_point(alpha = 0.3) +
  labs(title = "H2: Surprisal × Entropy Interaction Across
       x = "Block", y = "Interaction Coefficient", color =
  theme_minimal()
ggsave("plots/h3_interaction_by_block.png", p3, width = 8,
print(p3)

# Plot matrix size effect (averaging across blocks)
coef_summary <- coef_df %>%
  group_by(matrix_size, matrix_size_num) %>%
  summarize(
    surprisal_mean = mean(surprisal, na.rm = TRUE),
    surprisal_se = sd(surprisal, na.rm = TRUE) / sqrt(n()),
    entropy_mean = mean(prev_entropy, na.rm = TRUE),
    entropy_se = sd(prev_entropy, na.rm = TRUE) / sqrt(n())
    interaction_mean = mean(interaction, na.rm = TRUE),
    interaction_se = sd(interaction, na.rm = TRUE) / sqrt(n
  )

p4 <- ggplot(coef_summary, aes(x = matrix_size_num)) +
  geom_line(aes(y = surprisal_mean, color = "Surprisal")) +
  geom_point(aes(y = surprisal_mean, color = "Surprisal"))
  geom_errorbar(aes(ymin = surprisal_mean - surprisal_se,
                    ymax = surprisal_mean + surprisal_se, c
                width = 0.2) +
  geom_line(aes(y = entropy_mean, color = "Entropy")) +
  geom_point(aes(y = entropy_mean, color = "Entropy")) +
  geom_errorbar(aes(ymin = entropy_mean - entropy_se,
                    ymax = entropy_mean + entropy_se, color
                width = 0.2) +
  labs(title = "H3: Effect Size by Matrix Size (Averaged Ac
       x = "Matrix Size", y = "Coefficient Estimate", color
  theme_minimal()
ggsave("plots/h3_effects_by_matrix_size.png", p4, width = 8
print(p4)
```

Discussion

References

Farkas, B. C., Krajcsi, A., Janacsek, K., & Nemeth, D. (2024). The complexity of measuring reliability in learning tasks: An illustration using the Alternating Serial Reaction Time Task. *Behavior Research Methods*, *56*(1), 301–317. https:

//doi.org/10.3758/s13428-022-02038-5

Hedenius, M., Persson, J., Tremblay, A., Adi-Japha, E., Veríssimo, J., Dye, C. D., Alm, P., Jennische, M., Bruce Tomblin, J., & Ullman, M. T. (2011). Grammar predicts procedural learning and consolidation deficits in children with Specific Language Impairment. *Research in Developmental Disabilities*, *32*(6), 2362–2375. https://doi.org/10.1016/j.ridd.2011.07.026

Howard, D. V., Howard, J. H., Japikse, K., DiYanni, C., Thompson, A., & Somberg, R. (2004). Implicit Sequence Learning: Effects of Level of Structure, Adult Age, and Extended Practice. *Psychology and Aging*, *19*(1), 79–92. https://doi.org/10.1037/0882-7974.19.1.79

Howard Jr., J. H., & Howard, D. V. (1997). Age differences in implicit learning of higher order dependencies in serial patterns. *Psychology and Aging*, *12*(4), 634–656. https://doi.org/10.1037/0882-7974.12.4.634

Janacsek, K., Fiser, J., & Nemeth, D. (2012). The best time to acquire new skills: Age-related differences in implicit sequence learning across the human lifespan. *Developmental Science*, *15*(4), 496–505. https://doi.org/10.1111/j.1467-7687.2012.01150.x

Kassubek, J., Schmidtke, K., Kimmig, H., Lücking, C. H., & Greenlee, M. W. (2001). Changes in cortical activation during mirror reading before and after training: An fMRI study of procedural learning. *Cognitive Brain Research*, *10*(3), 207–217. https://doi.org/https://doi.org/10.1016/S0926-6410(00)00037-9

Kidd, E., & Arciuli, J. (2016). Individual Differences in Statistical Learning Predict Children's Comprehension of Syntax. *Child Development*, *87*(1), 184–193. https://doi.org/10.1111/cdev.12461

Lammertink, I., Boersma, P., Wijnen, F., & Rispens, J. (2017). Statistical Learning in Specific Language Impairment: A Meta-Analysis. *Journal of Speech, Language, and Hearing Research*, *60*(12), 3474–3486. https://doi.org/10.1044/2017_JSLHR-L-16-0439

Lum, J. A. G., Conti-Ramsden, G., Morgan, A. T., & Ullman, M. T. (2014). Procedural learning deficits in specific language impairment (SLI): A meta-analysis of serial reaction time task performance. *Cortex*, *51*, 1–10. https://doi.org/10.1016/j.cortex.2013.10.011

Lustig, C. (2022). *The transition from implicit to explicit knowledge representations in implicit sequence learning* [Text.thesis.doctoral, Universität zu Köln]. http://www.uni-koeln.de/

Mullaney, T. S. (2017). *The Chinese Typewriter: A History*. MIT Press.

Nissen, M. J., & Bullemer, P. (1987). Attentional requirements of learning: Evidence from performance measures. *Cognitive Psychology*, *19*(1), 1–32. https://doi.org/10.1016/0010-0285(87)90002-8

Oliveira, C. M., Hayiou-Thomas, M. E., & Henderson, L. M. (2023). The reliability of the serial reaction time task: Meta-analysis of test–retest correlations. *Royal Society Open Science*, *10*(7), 221542. https://doi.org/10.1098/rsos.221542

Song, S., Howard, J. H., & Howard, D. V. (2008). Perceptual sequence learning in a serial reaction time task. *Experimental Brain Research*, *189*(2), 145–158. https://doi.org/10.1007/s00221-008-1411-z

Squire, L. R. (2004). Memory systems of the brain: A brief history and current perspective. *Neurobiology of Learning and Memory*, *82*(3), 171–177. https://doi.org/https://doi.org/10.1016/j.nlm.2004.06.005

Ullman, M. T., & Pierpont, E. I. (2005). Specific Language Impairment is not Specific to Language: The Procedural Deficit Hypothesis. *Cortex*, *41*(3), 399–433. https://doi.org/10.1016/S0010-9452(08)70276-4

West, G., Vadillo, M. A., Shanks, D. R., & Hulme, C. (2018). The procedural learning deficit hypothesis of language learning disorders: We see some problems. *Developmental Science*, *21*(2), e12552. https://doi.org/10.1111/desc.12552