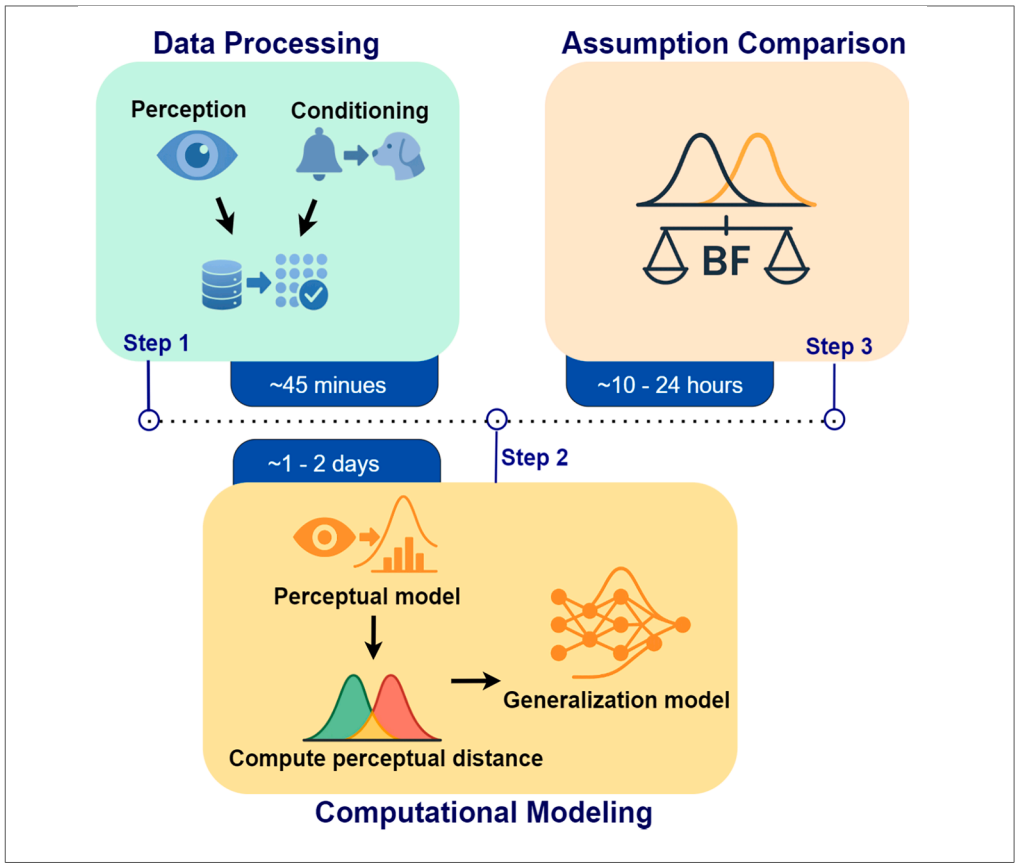
STAR Protocols

CellPress
OPEN ACCESS

Protocol

Computational protocol for hierarchical Bayesian modeling of perception and generalization in fear conditioning



Understanding human generalization behavior requires disentangling underlying cognitive and perceptual mechanisms. Here, we present a computational protocol to analyze individual differences in fear generalization by integrating a Bayesian state-space perceptual model with a hierarchical mixture generalization model. We describe steps for applying a state-space model to measure perceptual data and for calculating stimulus distance from these probabilistic representations. We then detail procedures for employing a hierarchical mixture generalization model to distinguish between perceptual and learning-based generalization processes.

Publisher's note: Undertaking any experimental protocol requires adherence to local institutional guidelines for laboratory safety and ethics.

Kenny Yu, Wolf Vanpaemel, Francis Tuerlinckx, Jonas Zaman

kenny.yu@kuleuven.be

Highlights

Steps for applying a Bayesian state-space model to track perceptual representations

Procedures for computing stimulus similarity from perceptual distribution overlap

Instructions for fitting a hierarchical mixture model to identify generalization pathways

Guidance on comparing perceptual assumptions using Bayes factors in a super-model

Yu et al., STAR Protocols 7, 104400
March 20, 2026
https://doi.org/10.1016/j.xpro.2026.104400

STAR Protocols

CellPress
OPEN ACCESS

Protocol

Computational protocol for hierarchical Bayesian modeling of perception and generalization in fear conditioning

Kenny Yu,[1,4,5,*] Wolf Vanpaemel,[1] Francis Tuerlinckx,[1,3] and Jonas Zaman[1,2,3]

[1]Quantitative Psychology and Individual Differences, KU Leuven, 3000 Leuven, Belgium

[2]REVAL Rehabilitation Research, Faculty of Rehabilitation Sciences, UHasselt, 3590 Diepenbeek, Belgium

[3]These authors contributed equally

[4]Technical contact

[5]Lead contact

*Correspondence: kenny.yu@kuleuven.be
https://doi.org/10.1016/j.xpro.2026.104400

SUMMARY

Understanding human generalization behavior requires disentangling underlying cognitive and perceptual mechanisms. Here, we present a computational protocol to analyze individual differences in fear generalization by integrating a Bayesian state-space perceptual model with a hierarchical mixture generalization model. We describe steps for applying a state-space model to measure perceptual data and for calculating stimulus distance from these probabilistic representations. We then detail procedures for employing a hierarchical mixture generalization model to distinguish between perceptual and learning-based generalization processes.

BEFORE YOU BEGIN

This section prepares researchers for implementing the computational protocol. We begin with a conceptual overview of the modeling framework, followed by methodological foundations, software requirements, data specifications, and reproducibility practices.

Innovation

This protocol advances fear generalization research through three methodological innovations.

First, we model perception as dynamic probability distributions rather than static point estimates, using a Bayesian state-space framework with Kalman filter dynamics that captures how perceptual uncertainty evolves across trials.

Second, we introduce a distribution-overlap metric for computing stimulus similarity, replacing traditional Euclidean distances with a measure that reflects perceptual confusability arising from uncertain representations.

Third, we integrate these perceptual distributions into a hierarchical mixture generalization model that probabilistically assigns individuals to distinct pathways (Non-learners, Overgeneralizers, Physical Generalizers, and Perceptual Generalizers), enabling researchers to disentangle whether broad generalization stems from liberal associative transfer or genuine perceptual uncertainty.

Unlike previous approaches that inferred perceptual distributions from generalization behavior itself, our framework derives them independently from trial-by-trial perceptual data, avoiding theoretical circularity. The implementation uses open-source tools (R, JAGS) with documented code.

Conceptual overview of the modeling framework

Traditional fear generalization research treats stimulus similarity as fixed and identical across individuals. If two stimuli are physically distinct, they are assumed to be perceived as distinct by all observers. This assumption obscures two important sources of individual variation: differences in how stimuli are perceived, and differences in how learned associations transfer to novel stimuli.

From aggregate gradients to individual mechanisms

Yu et al. (2023)[1] introduced a hierarchical mixture modeling framework that addresses both sources. The mixture structure recognizes that what appears as a single "generalization gradient" in aggregate data actually conflates qualitatively distinct processes.

The model probabilistically assigns each participant to one of four pathways: (1) Non-learners, for whom conditioning failed to establish the association; (2) Overgeneralizers, who transfer learned associations broadly regardless of stimulus similarity; (3) Physical Generalizers, whose responses depend on physical stimulus similarity; and (4) Perceptual Generalizers, whose responses depend on perceptual similarity. This framework incorporated individual perceptual measurements, but treated them as point estimates.

From point estimates to probability distributions

Yu et al. (2025)[2] extends this framework by modeling perception as probability distributions rather than points. Using a Bayesian state-space model with Kalman filter dynamics, each observer's perceptual representation is characterized by both a mean (their best estimate) and variance (their uncertainty), updated recursively across trials. This distributional representation transforms how stimulus similarity is computed: rather than distance between point estimates, similarity becomes the overlap between uncertain distributions. Consequently, perceptual imprecision (not just perceptual bias) contributes to stimulus confusability.

The theoretical test

Integrating these components enables a critical distinction. Two individuals may exhibit identically broad generalization for fundamentally different reasons: one because associative learning transfers liberally, the other because perceptual uncertainty renders stimuli genuinely confusable.

By independently estimating perceptual distributions and using their overlap as input to the mixture generalization model, this protocol disentangles these mechanisms. Researchers can now ask not only who generalizes broadly, but why. For a review of this confound, see Zaman et al. (2021)[3]; for discussion of how perceptual and cognitive processes interact, see Yu et al. (2025b).[4]

Methodological foundations

Confirm familiarity with the following methodological areas before beginning the computational analysis.

Bayesian inference principles

Understanding Bayesian inference is essential for implementing and interpreting this protocol.

Core framework. Likelihood functions, prior distributions, and posterior distributions combine through Bayes' theorem to enable parameter estimation and uncertainty quantification. Unlike frequentist approaches that yield point estimates and confidence intervals, Bayesian inference produces full probability distributions over parameters, directly expressing uncertainty about parameter values given the observed data.

Prior specification. Priors encode knowledge or assumptions about parameters before observing data. This protocol uses weakly informative priors that regularize estimation without strongly constraining results. Understanding the distinction between informative, weakly informative, and noninformative priors helps in adapting the protocol to different contexts.

Posterior interpretation. The posterior distribution represents updated beliefs about parameters after observing data. Interpreting posteriors involves examining central tendency (mean, median, mode), spread (standard deviation, credible intervals), and shape (skewness, multimodality).

Markov chain Monte Carlo (MCMC). MCMC methods generate samples that approximate the posterior because posterior distributions are often analytically intractable. Understanding that MCMC produces dependent samples that require convergence assessment is crucial for valid inference.

For comprehensive introductions to Bayesian statistics, see Gelman et al. (2020),[5] Wagenmakers et al. (2018),[6] and Kruschke (2010).[7]

Bayesian model comparison

This protocol uses Bayes factors to compare competing models of perceptual processing.

Bayes factors. Bayes factors quantify the relative evidence that data provide for one model versus another, expressed as a ratio of marginal likelihoods. A Bayes factor of 10 indicates the data are 10 times more likely under one model than the other.

Savage Dickey method. The protocol implements the Savage Dickey density ratio method for computing Bayes factors within nested models. This approach compares the posterior density at a point of interest (e.g., the null hypothesis value) to the prior density at that same point.

For background on Bayes factors and the Savage Dickey method, see Wagenmakers et al. (2010).[8]

Reporting Bayesian analyses

Transparent reporting of Bayesian analyses ensures reproducibility and facilitates scientific communication. Researchers should familiarize themselves with established reporting guidelines before conducting analyses. Kruschke (2021)[9] provides comprehensive recommendations for reporting Bayesian analyses in the behavioral sciences. Key reporting elements include:

Prior specification transparency. Report all prior distributions with their parameterization and justification, ensuring priors in the JAGS model files are fully documented with rationale explained.

MCMC diagnostics. Report convergence diagnostics including the $\widehat{R}$ statistic (values below 1.01 indicate convergence), effective sample size (ESS; larger values indicate less autocorrelation), and visual inspection of trace plots.

Posterior summaries. Report posterior distributions using multiple summary statistics including a measure of central tendency, a measure of uncertainty, and credible interval bounds (typically 95%).

Credible intervals. Use highest density intervals (HDIs) when posteriors are asymmetric, or equal tailed intervals (ETIs) when posteriors are approximately symmetric, stating which type is reported.

Sensitivity analyses. Report how conclusions change under alternative prior specifications, which is particularly important for Bayes factors.

State-space modeling

The perceptual model employs state space modeling principles to track latent perceptual states over time. State space models distinguish between observable measurements and unobservable (latent) states that evolve according to dynamic equations. Key concepts include:

State equation. Describes how the latent state (perception) evolves from one trial to the next.

Observation equation. Links the latent state to observed measurements (perceptual ratings).

Kalman filtering. An algorithm that optimally estimates latent states by recursively combining predictions with observations, weighting each by their respective uncertainties.

The protocol explains implementation details, but conceptual familiarity with dynamic systems aids interpretation. For an accessible introduction to Kalman filtering, see the comprehensive tutorial at https://www.kalmanfilter.net/default.aspx.

Hierarchical (multilevel) modeling

Hierarchical models estimate parameters at multiple levels simultaneously, capturing both group-level regularities and individual-level variation.

Structure. Individual participants have unique parameter values (e.g., their own learning rate). These individual parameters are drawn from population-level distributions. The model estimates both individual parameters and population distribution parameters (hyperparameters).

Partial pooling. Extreme individual estimates are shrunk toward the group mean, reducing the influence of noise while preserving genuine individual differences.

Population inference. The structure provides population-level inferences that generalize beyond the specific sample studied.

For an overview of hierarchical modeling in psychology, see Lee (2011).[10]

Software environment and technical setup

1. R Statistical Environment.
   a. Install R version 4.1.1 or later from https://cran.r-project.org/.

   Note: RStudio (https://posit.co/products/open-source/rstudio/) is recommended for an enhanced development experience, including syntax highlighting, integrated help, and project management.

2. Required R packages.
   a. Install necessary packages for this protocol:

```
# Install all required packages

required_packages <- c("jagsUI", "logspline", "data.table", "dplyr",

            "tidyr", "ggplot2", "bayesplot", "coda",

            "patchwork")

install.packages(required_packages)
```

b. Package purposes.

Note: `jagsUI` provides the JAGS interface for running JAGS models from R; `logspline` enables density estimation for Savage Dickey Bayes factor computation; `data.table`, `dplyr`, and `tidyr` handle data manipulation; `ggplot2`, `bayesplot`, and `patchwork` provide visualization capabilities; and `coda` supports MCMC diagnostics for convergence assessment.

3. JAGS (Just Another Gibbs Sampler).
   a. Install JAGS version 4.3.1 or later from http://mcmc-jags.sourceforge.net/.

Note: `JAGS` is a standalone program that performs MCMC sampling; `R` interfaces with `JAGS` through the `jagsUI` package.

Note: Knowledge of JAGS syntax is essential for understanding and adapting the model specifications. Consult the JAGS User Manual (available at https://sourceforge.net/projects/mcmc-jags/files/Manuals/) for syntax reference and modeling conventions.

4. Installation verification.
   a. Test `JAGS` and `jagsUI` configuration with a simple model:

```
# Load required library
library(jagsUI)
# Simple test model
model_string <- "
model {
  for(i in 1:N) {
    y[i] ~ dnorm(mu, tau)
  }
  mu ~ dnorm(0, 0.01)
  tau ~ dgamma(0.01, 0.01)
  sigma <- 1/sqrt(tau)
}"
# Test data
set.seed(123)
test_data <- list(y = rnorm(10), N = 10)
# Test model compilation and sampling
test_model <- jags(data = test_data,
         model.file = textConnection(model_string),
         parameters.to.save = c("mu", "sigma"),
         n.chains = 2,
         n.iter = 1000,
         n.burnin = 500,
         n.thin = 1)
# Verify convergence
```

```
if(max(test_model$Rhat) < 1.1) {
  print("JAGS and jagsUI installation successful!")
  print(test_model$summary)
} else {
  warning("Installation may have issues. Check JAGS configuration.")
}
```

Note: If installation fails, ensure JAGS is in the system PATH and restart R/RStudio after installation. See the Troubleshooting section for common installation issues.

Note: While `JAGS` is recommended for this protocol, the models can be adapted for `Stan` (`rstan`/`cmdstanr`) or `PyMC` (`Python`). Model syntax translation is required, with particular attention to prior specifications. For `Stan` users specifically: this protocol uses uniform priors on several parameters, which work well in `JAGS` (Gibbs sampling) but are discouraged in `Stan` because Hamiltonian Monte Carlo relies on gradient information that is undefined at hard boundaries. When adapting to `Stan`, replace uniform priors with weakly informative alternatives (e.g., half-normal or half-*t* distributions for variance parameters) to avoid divergent transitions.

Computational requirements

This protocol involves computationally intensive hierarchical Bayesian models with MCMC sampling.

Reference dataset and timing

The timing estimates below are based on the reference dataset from Yu et al. (2025a),[2] which comprises 40–80 participants with approximately 180 trials per participant, yielding 7,200–14,400 total trial-level observations. With this dataset size on a modern workstation (8+ cores, 16+ GB RAM), the Bayesian perceptual model requires 4–8 hours, the hierarchical mixture generalization models (two versions) require 10–24 hours, and the super-model comparison requires 12–24 hours.

Scaling with data size

Computation time increases with both the number of participants (N) and trials (T). The *per-iteration cost* scales approximately with $N \times T$ (the total number of observations), and the mixture generalization model has additional overhead from discrete group assignment computations. The *convergence requirements* may increase with data complexity; larger N introduces more individual-level parameters and can create multimodal posterior landscapes that require extended sampling.

Key considerations

Total wall-clock time depends on both per-iteration cost and convergence requirements. Parallel processing across MCMC chains is recommended (4 chains on 4 cores is standard). Memory usage increases with the number of participants (N), the number of trials (T), and the number of monitored parameters. Models scale nonlinearly with data complexity due to hierarchical parameter dependencies, and convergence may require extended sampling periods for complex posteriors, particularly with larger N where individual differences create multimodal landscapes.

Practical guidance for different dataset sizes:

Dataset size	$N \times T$	Relative time	Example
Small	<5,000	0.5–0.7×	30 participants, 150 trials
Reference	7,000–15,000	1×	40–80 participants, 180 trials
Large	15,000–30,000	1.5–2×	100 participants, 200 trials
Very large	>30,000	2–4×	150+ participants, 200+ trials

For datasets substantially larger than the reference (e.g., >100 participants), consider using high-performance computing (HPC) clusters, reducing the number of MCMC iterations initially to verify model specification, or fitting subsets of participants to estimate timing before full analysis.

Hardware recommendations

The recommended minimum configuration is 4 cores with 8 GB RAM (suitable for small datasets only); 4 cores allows running 4 MCMC chains in parallel, though fewer cores can be used with sequential chain execution. The optimal configuration is 8+ cores with 16–32 GB RAM. For large datasets, consider HPC clusters or cloud computing (AWS, Google Cloud, Azure).

Experimental paradigm and terminology

This protocol is designed for data from Pavlovian fear conditioning experiments,[11] a foundational paradigm for studying associative learning.

In this paradigm, an initially neutral stimulus (the conditioned stimulus, CS) is repeatedly paired with an aversive outcome (the unconditioned stimulus, US), such as a mild electric shock or loud noise. Through repeated pairings, the CS acquires the ability to elicit a defensive response (the conditioned response, CR), such as increased skin conductance, startle reflex, or subjective fear ratings.

In simple conditioning, a single CS is paired with the US (denoted CS+). In differential conditioning, one stimulus is paired with the US (CS+) while another is explicitly unpaired (CS-), allowing assessment of discriminative learning.

A typical fear generalization experiment proceeds in two phases. In the *acquisition phase*, the CS–US association is established through repeated pairings, and participants learn to expect the US following CS+ presentation. In the *generalization phase*, novel stimuli (test stimuli, TS) that vary in similarity to the CS+ are presented without reinforcement, and the degree to which the conditioned response transfers to these novel stimuli indexes generalization behavior. See Yu et al. (2023)[1] for a validated implementation of this experimental design with continuous perceptual and expectancy measurements.

Data requirements

Required measurements

Experiments must include continuous perceptual ratings of stimulus features (e.g., size, color intensity) on each trial, continuous expectancy ratings (e.g., US expectancy) on each trial, trial-by-trial recording of stimulus identity and reinforcement outcomes, and clear identification of experimental phase (acquisition vs. generalization).

Data structure

Each row represents a single participant-trial observation. Required variables: `participant_id`, `trial_number`, `stimulus_id`, `perceptual_rating`, `expectancy_rating`, `reinforcement` (0/1), `phase` (acquisition/generalization).

Model specification and validation knowledge

Prior specification philosophy

This protocol uses weakly informative priors to reflect limited prior knowledge about model parameters while providing regularization. For future studies, researchers may consider incorporating domain knowledge through more informative priors or using posterior distributions from Yu et al. (2025a)[2] as priors in subsequent research to build cumulative knowledge.

Parameter interpretation

Understanding the psychological meaning of key model parameters is essential for meaningful interpretation. For comprehensive theoretical background, see Yu et al. (2025a)[2] and Yu et al. (2023).[1]

Perceptual model parameters. β_0 and β_1 (sensory mapping parameters transforming physical stimuli to perceptual estimates), ω (process noise forgetting rate determining how quickly perceptual uncertainty decays over trials, with higher values indicating faster stabilization), η (initial uncertainty magnitude governing perceptual precision at experiment onset), and σ_{sensory} (individual sensory mapping uncertainty reflecting measurement noise).

Generalization model parameters. α (learning rate governing speed of CS–US association acquisition), λ (generalization rate controlling the steepness of similarity-dependent fear decay), and w_0, w_1 (response scaling parameters mapping latent fear strength to observed ratings).

Mixture model parameters. π (group allocation probabilities indicating population prevalence of each pathway) and group-specific constraints defining Nonlearners, Overgeneralizers, Physical Generalizers, and Perceptual Generalizers.

MCMC convergence assessment

Knowledge of MCMC diagnostics is essential for valid inference. Key diagnostics include:

$\widehat{R}$ statistic. Compares within-chain and between-chain variance. Values below 1.01 indicate good convergence; values between 1.01 and 1.1 suggest results may be usable but warrant closer inspection and additional iterations may improve precision; values above 1.1 indicate clear problems requiring extended sampling.[5,12,13]

Effective sample size (ESS). Estimates the number of independent samples accounting for autocorrelation. Higher ESS yields more precise posterior estimates.

Trace plots. Visual inspection of parameter chains should show good mixing (chains exploring the same region) without trends or stuck periods.

Model validation and sensitivity analysis

Bayesian workflow involves multiple validation stages that assess different aspects of model adequacy. This protocol employs three complementary approaches.

Prior predictive checking (before model fitting). Evaluates whether the chosen priors encode reasonable assumptions by simulating data from the prior distributions alone, without conditioning on observed data. This check asks: "Before seeing any data, what range of outcomes does my model consider plausible?" Priors that generate implausible predictions (e.g., negative learning rates would cause fear responses to decrease rather than increase following reinforcement) should be revised before fitting.

Posterior predictive checking (after model fitting). Assesses whether the fitted model can reproduce key features of the observed data. This check asks: "Does my fitted model capture the structure of the observed data?"

This involves generating simulated datasets from the posterior distribution, comparing summary statistics (means, variances, quantiles) between simulated and observed data, and identifying systematic discrepancies. A model that fits well should generate simulated data that is statistically indistinguishable from the observed data on relevant summary measures.

Prior sensitivity analysis (after model fitting)

Examines whether substantive conclusions depend on specific prior choices. This involves refitting the model under alternative prior specifications (e.g., wider or narrower priors, different distributional families) and assessing whether key inferences (such as group allocation probabilities or learning rate estimates) remain stable. Conclusions that change dramatically under reasonable alternative priors warrant caution and should be reported transparently.

For comprehensive background on Bayesian workflow, diagnostics, and model checking, see Gelman et al. (2020).[14]

Reproducibility and documentation practices

The following practices ensure reproducibility and are applied throughout this protocol.

Version control

Use Git for tracking analysis changes and GitHub for collaboration and transparency. Version control enables tracking all modifications to analysis code, reverting to previous versions if needed, documenting the rationale for changes through commit messages, and sharing complete analysis history with collaborators and reviewers.

Literate programming

Create reproducible workflows using R Markdown or Quarto documents that integrate narrative descriptions of analytical decisions, executable code chunks, results and visualizations, and interpretation and conclusions. This approach produces self-documenting analyses where the final report contains all information needed for reproduction.

Seed management

Set fixed random seeds for all stochastic procedures to ensure fully reproducible results:

Fixed seed for reproducibility (document this value in reports).

set.seed(2025).

Use a consistent seed throughout the project and document the seed value in analysis reports, manuscripts, and alongside saved output files. Fixed seeds ensure that anyone running the code will obtain identical results regardless of when the analysis is performed. This prevents inadvertent changes in stochastic components of the workflow. This prevents inadvertent changes in stochastic components of the workflow[X].

Project organization

Maintain a systematic file structure separating raw data, processing scripts, models, and outputs.

```
project_name/
|-- data/
| |-- raw/ # Original data files (never modified)
| +-- processed/ # Cleaned, analysis-ready data
|-- scripts/
| |-- 01_data_preprocessing.R
| |-- 02_perceptual_model.R
| |-- 03_distance_computation.R
| |-- 04_generalization_model.R
| |-- 05_model_comparison.R
| +-- 06_results_visualization.R
|-- models/
```

```
| |-- jags_models/ # JAGS model specification files (.txt)
| +-- fitted_models/ # Saved model objects (.RData, .rds)
|-- outputs/
| |-- figures/ # Publication ready visualizations
| |-- tables/ # Summary tables
| +-- diagnostics/ # Convergence plots, diagnostic reports
|-- docs/
| |-- analysis_report.Rmd # Main reproducible analysis document
| +-- supplementary/ # Additional documentation
+-- README.md # Project overview and instructions
```

This structure facilitates collaboration, review, and long-term archiving.

KEY RESOURCES TABLE

REAGENT or RESOURCE	SOURCE	IDENTIFIER
Software and algorithms		
R Statistical Software (version 4.1.1 or higher)	R Foundation for Statistical Computing	RRID:SCR_001905; https://www.r-project.org/
JAGS (Just Another Gibbs Sampler) version 4.3.1 or higher	Martyn Plummer	RRID:SCR_017274; https://mcmc-jags.sourceforge.io/
R package: jagsUI	CRAN	https://cran.r-project.org/package=jagsUI
Deposited data		
Protocol materials and example data	Open Science Framework	https://osf.io/a5xyg/
Other		
RStudio IDE (recommended)	Posit Software, PBC	RRID:SCR_000432; https://posit.co/products/open-source/rstudio/

STEP-BY-STEP METHOD DETAILS

In this section, we provide a step-by-step workflow for the analysis (see Figure 1 for an overview).

Data processing

Timing: 30–45 min

Note: Transform fear conditioning data into JAGS-compatible matrix format following the computational workflow (see Figure 1). Data processing splits into two parallel streams: perceptual data for the Bayesian perceptual model and generalization data for the hierarchical mixture generalization model. JAGS requires rectangular matrix structures where rows represent participants and columns represent trials.

1. Load all required R packages for this protocol.

```
# Load all required packages
packages <- c("jagsUI", "logspline", "data.table", "dplyr",
       "tidyr", "ggplot2", "bayesplot", "coda", "patchwork")
invisible(lapply(packages, library, character.only = TRUE))
```

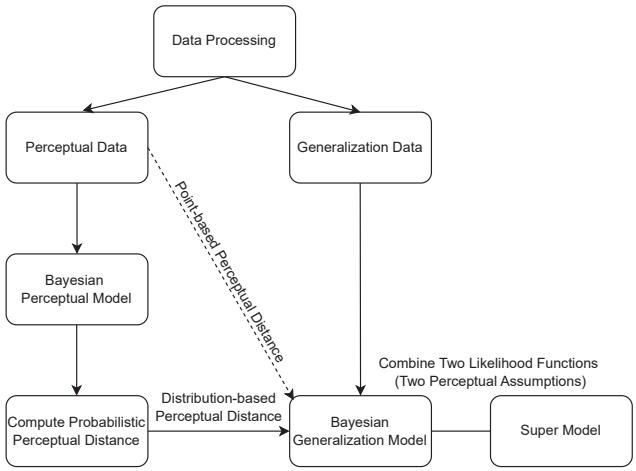


Figure 1. The workflow analyzes perceptual and generalization data through four sequential stages

(1) Bayesian perceptual modeling to estimate individual stimulus-to-perception transformations with uncertainty dynamics, (2) computation of three distance measures (physical, raw perceptual, and model-based), (3) Bayesian hierarchical mixture generalization modeling to identify individual learning and generalization mechanisms, and (4) super-model comparison using Bayes factors to determine whether distribution-based or point-based perceptual assumptions better explain fear generalization behavior.

2. Prepare trial-by-trial perceptual ratings for dynamic state-space modeling.
   a. Set up perceptual model matrices:

```
# Load raw experimental data in tidy format
# Required columns: participant_id, trial_number, stimulus_id,
# perceptual_rating, expectancy_rating, reinforcement, phase
# Get basic dimensions
participant_ids <- unique(raw_data$participant_id)
n_participants <- length(participant_ids)
n_trials <- max(raw_data$trial_number)
# Create perceptual data matrices
perceptual_responses <- matrix(NA, n_participants, n_trials)
stimulus_indices <- matrix(NA, n_participants, n_trials)
# Create stimulus mapping from stimulus_id to numeric indices
stimulus_values <- sort(unique(raw_data$stimulus_id))
n_stimuli <- length(stimulus_values)
# Fill matrices from experimental data
for (i in 1:n_participants) {
  participant_data <- subset(raw_data,
              participant_id == participant_ids[i])
  # Fill perceptual responses
  perceptual_responses[i, participant_data$trial_number] <-
    participant_data$perceptual_rating
  # Fill stimulus indices (convert to integers 1,2,3...)
```

```
  stimulus_indices[i, participant_data$trial_number] <-
  match(participant_data$stimulus_id, stimulus_values)
}
```

3. Prepare expectancy ratings and experimental indicators for generalization modeling.
   a. Set up generalization model matrices:

```
# Create generalization data matrices
expectancy_responses <- matrix(NA, n_participants, n_trials)
cs_indicators <- matrix(NA, n_participants, n_trials)
reinforcement_outcomes <- matrix(NA, n_participants, n_trials)
# Fill matrices from experimental data
for (i in 1:n_participants) {
  participant_data <- subset(raw_data,
            participant_id == participant_ids[i])
  # Fill expectancy responses
  expectancy_responses[i, participant_data$trial_number] <-
    participant_data$expectancy_rating
  # Fill CS indicators (1 = CS trial, 0 = test trial)
  cs_indicators[i, participant_data$trial_number] <-
    as.numeric(participant_data$phase == "acquisition")
  # Fill reinforcement outcomes
  reinforcement_outcomes[i, participant_data$trial_number] <-
    participant_data$reinforcement
}
```

4. Create tracking matrices for distance computation.

```
# Create trial type and stimulus sequence matrices
trial_type <- matrix(NA, n_participants, n_trials)
stimulus_sequence <- matrix(NA, n_participants, n_trials)
cs_assignment <- rep(NA, n_participants)
for (i in 1:n_participants) {
  participant_data <- subset(raw_data,
              participant_id == participant_ids[i])
  # Fill trial type matrix
  trial_type[i, participant_data$trial_number] <-
    ifelse(participant_data$phase == "acquisition", "CS", "test")
```

```
  # Fill stimulus sequence (indices into stimulus_values)
  stimulus_sequence[i, participant_data$trial_number] <-
    match(participant_data$stimulus_id, stimulus_values)
  # Determine CS assignment for this participant
  cs_trials <- participant_data[
    participant_data$phase == "acquisition", ]
  if (nrow(cs_trials) > 0) {
    cs_assignment[i] <- match(cs_trials$stimulus_id[1],
                stimulus_values)
  }
}
```

5. Save all prepared data matrices with exact variable names.

```
save(
  # Perceptual data variables
  perceptual_responses, stimulus_indices, stimulus_values,
  n_participants, n_trials, n_stimuli,
  # Generalization data variables
  expectancy_responses, cs_indicators, reinforcement_outcomes,
  # Distance calculation variables
  trial_type, stimulus_sequence, cs_assignment,
  file = "data/processed/prepared_data_matrices.RData"
)
```

6. Verify the required data structure format.
   a. Example processed matrices (3 participants, 4 trials):

```
PERCEPTUAL DATA:
perceptual_responses:
               Trial1  Trial2  Trial3  Trial4
Participant1:  72      68      51      49
Participant2:  69      NA      53      50
Participant3:  70      65      48      47
stimulus_indices:
               Trial1  Trial2  Trial3  Trial4
Participant1:  4       4       1       2
Participant2:  4       4       1       2
Participant3:  4       4       1       2
```

```
GENERALIZATION DATA:

expectancy_responses:

               Trial1  Trial2  Trial3  Trial4

Participant1:  8       7       4       3

Participant2:  9       8       5       4

Participant3:  7       6       3       2

cs_indicators:

               Trial1  Trial2  Trial3  Trial4

Participant1:  1       1       0       0

Participant2:  1       1       0       0

Participant3:  1       1       0       0

reinforcement_outcomes:

               Trial1  Trial2  Trial3  Trial4

Participant1:  1       0       0       0

Participant2:  0       1       0       0

Participant3:  1       1       0       0
```

b. Key data structure requirements:

Note: All matrices have dimensions [n_participants × n_trials]. Variable names match JAGS data list requirements in Bayesian perceptual model and hierarchical mixture generalization model.

Note: The perceptual data stream feeds into the Bayesian perceptual model, while generalization data supports the hierarchical mixture generalization model.

Perceptual model

Timing: 4–8 h

Note: This section implements a dynamic perceptual model that captures how individuals transform physical stimuli into probabilistic mental representations over time. The model processes perceptual data from Data processing to generate trial-by-trial perceptual distributions, which will be used to compute stimulus similarity in Stimulus distance computation.

7. Understand the theoretical foundation and modeling approach.

Note: Contemporary theories characterize perception as probabilistic inference.[15–17] In a full Bayesian observer model, a physical stimulus s generates a stochastic internal measurement $m \sim p(m|s)$, which the observer combines with prior beliefs to compute a posterior distribution: $p(s|m) \propto p(m|s) \cdot p(s)$. Full implementations model the sensory measurement as a latent variable requiring marginalization, and include a separate decision stage with its own noise parameters.

Note: The goal of this protocol is to derive perceptual distributions for computing stimulus similarity via Gaussian overlap (see Stimulus distance computation). This downstream

application depends only on the mean and standard deviation of each distribution. Any information beyond these statistics is irrelevant for computing overlap, which motivates a simplified approach. Rather than implementing a full Bayesian observer with latent sensory measurements, the model in Yu et al. (2025)[2] directly tracks the evolution of perceptual estimates (μ_ψ,σ_ψ) over trials using a Kalman-like recursive update. Sensory noise is estimated as a separate individual-difference parameter $\sigma_{sensory,i}$, while responses are sampled directly from the perceptual distribution without a separate decision stage, thus conflating any decision noise with perceptual uncertainty. Researchers testing hypotheses about the separation of perceptual and decisional processes should consider fuller implementations.[17]

Note: The state transition follows a specific functional form: the perceptual distribution from trial j informs the prior for trial j+1, with added process noise that decays exponentially as $\eta \cdot exp(-\omega_i j)$. This functional form was validated for fear generalization contexts in Yu et al. (2025).[2] Researchers working with longer time scales, multi-dimensional stimuli, or contexts where perceptual representations might drift should consider alternative state transition models.

8. Understand the model specification and key equations.

Note: The model implements a state-space framework with Kalman filter dynamics. At each trial j for participant i, the model: (1) computes the sensory estimate for the current stimulus, (2) combines this with the prior via a weighted update, (3) generates a response, and (4) transitions to the next trial with added process noise.

Note: *Sensory transformation equation.* Physical stimulus values are transformed into sensory estimates through an individual-specific sigmoid function:

$$\mu_{sensory,ik} = \frac{U}{1+\exp\left(-\left(\beta_{0,i}+\beta_{1,i}\cdot x_k\right)\right)} \quad \text{(Equation 1)}$$

where U=200 is the upper bound of the rating scale, x_k is the physical value of stimulus k, and $\beta_{0,i}$ (intercept) and $\beta_{1,i}$ (slope) are individual-specific parameters. The sigmoid bounds outputs to [0,U] while allowing flexible mappings, from compressed (low β_1) to expanded (high β_1). The sensory estimate has associated uncertainty $\sigma_{sensory,i}$, an individual-difference parameter reflecting encoding precision. Empirical support for this transformation comes from several recent studies using the same measurement paradigm.[1,2,18,19]

Note: *Initialization.*

$$\mu_{\psi,i1} = \mu_{sensory,i,s[1]} \quad \text{(Equation 2)}$$

$$\sigma^2_{\psi,i1} = 10000 \quad \text{(Equation 3)}$$

The initial mean is set to the sensory estimate of the first stimulus. The large initial variance (equivalent to SD = 100) ensures that the Kalman gain on the first trial approaches 1, so initial perception is driven primarily by sensory evidence rather than the (uninformative) prior.

Note: *Kalman gain equation (Bayesian updating).* The Kalman gain determines the weighting of sensory evidence versus prior expectations based on their relative uncertainties:

$$K_{i,j+1} = \frac{\sigma^2_{\psi,ij}}{\sigma^2_{\psi,ij}+\sigma^2_{sensory,i}} \quad \text{(Equation 4)}$$

where $\sigma^2_{\psi,ij}$ is the current perceptual variance and $\sigma^2_{\text{sensory},i}$ is the sensory variance. When perceptual variance is high relative to sensory variance, K approaches 1 and the update relies heavily on sensory input. When perceptual variance is low, K approaches 0 and the estimate changes little.

Note: *Perceptual mean update equation.*

$$\mu_{\psi,i(j+1)} = \mu_{\psi,ij} + K_{i,j+1} \cdot \left(\mu_{\text{sensory},i,s[j+1]} - \mu_{\psi,ij}\right) \quad \text{(Equation 5)}$$

where $s[j+1]$ indexes the stimulus presented on trial $j+1$. The new estimate is a weighted average of prior expectation ($\mu_{\psi,ij}$) and sensory input (μ_{sensory}), with the weight determined by their relative variances.

Note: *Variance update equation (uncertainty dynamics).*

$$\sigma^2_{\psi,i(j+1)} = \left(1 - K_{i,j+1}\right) \cdot \sigma^2_{\psi,ij} + \eta \cdot \exp\left(-\omega_i \cdot j\right) \quad \text{(Equation 6)}$$

The first term reflects variance reduction from incorporating sensory information. The second term adds process noise that decays exponentially over trials, preventing variance from collapsing to zero and capturing the pattern that perception becomes more stable with experience. The decay rate ω_i is individual-specific; the initial magnitude η is a population parameter.

Note: *Response generation equation.*

$$\psi_{ij} \sim \mathcal{N}\left(\mu_{\psi,ij}, \sigma^2_{\psi,ij}\right) \quad \text{(Equation 7)}$$

Responses are sampled from the current perceptual distribution, reflecting both the observer's best estimate (the mean) and their uncertainty (the variance).

Parameter	Interpretation	Constraint	Level
$\beta_{0,i}$	Sigmoid intercept (horizontal shift)	$\mathbb{R}$	Individual
$\beta_{1,i}$	Sigmoid slope (perceptual sensitivity)	>0	Individual
$\sigma_{\text{sensory},i}$	Sensory encoding noise (SD)	>0	Individual
ω_i	Process noise decay rate	>0	Individual
η	Initial process noise magnitude	>0	Population

9. Review parameter summary.
   a. Reference the parameter table:
   b. Specify priors and validate them.
      i. Specify hierarchical structure:
      Individual parameters are drawn from population distributions:

$$\beta_{0,i} \sim \mathcal{N}\left(\mu_{\beta_0}, \sigma^2_{\beta_0}\right) \quad \text{(Equation 8)}$$

$$\beta_{1,i} \sim \text{LogNormal}\left(\mu_{\beta_1}, \sigma^2_{\beta_1}\right) \quad \text{(Equation 9)}$$

$$\sigma_{\text{sensory},i} \sim \text{LogNormal}\left(\mu_{\sigma_s}, \sigma^2_{\sigma_s}\right) \quad \text{(Equation 10)}$$

$$\omega_i \sim \text{LogNormal}\left(\mu_{\omega}, \sigma^2_{\omega}\right) \quad \text{(Equation 11)}$$

      Log-normal distributions ensure positivity for parameters constrained to be positive.
      ii. Specify population-level hyperpriors (calibrated for 0–200 mm rating scale):

$$\mu_{\beta_0} \sim \mathcal{N}\left(0, 5^2\right) \sigma_{\beta_0} \sim \text{half} - t(2) \quad \text{(Equation 12)}$$

$$\mu_{\beta_1} \sim \mathcal{N}(-3, 2^2)\sigma_{\beta_1} \sim \text{half} - t(2) \quad \text{(Equation 13)}$$

$$\mu_{\sigma_s} \sim \mathcal{N}(0, 2^2)\sigma_{\sigma_s} \sim \text{half} - t(2) \quad \text{(Equation 14)}$$

$$\mu_{\omega} \sim \mathcal{N}(-5, 5^2)\sigma_{\omega} \sim \text{half} - t(2) \quad \text{(Equation 15)}$$

$$\eta \sim \text{LogNormal}(2, 1) \quad \text{(Equation 16)}$$

The half-t(2) priors on population SDs are implemented via precision parameterization: $\tau \sim$Scaled-Gamma(2,1) and $\sigma = 1/\sqrt{\tau}$[20].

These weakly informative priors were validated in Yu et al. (2025).[2] A prior sensitivity analysis (see supplementary materials of Yu et al. (2025)[2]) confirmed that posterior estimates of the perceptual distributions are robust across plausible prior specifications.

iii. Conduct prior predictive checking (before model fitting):

Note: This checks whether the priors are reasonable for the problem domain. Simulate data from the prior and verify it generates plausible perceptual trajectories: responses within the rating scale, decreasing uncertainty over trials, etc. If most simulations produce implausible patterns, the priors need adjustment before fitting.

```
# Prior predictive simulation
set.seed(123)
n_sim <- 100
n_trials <- 50
U <- 200
stim_values <- seq(50, 120, by = 10) # Physical stimulus values
results <- data.frame(
  sim = integer(), prop_in_range = numeric(),
  mean_response = numeric(), var_decreases = logical()
)
for (s in 1:n_sim) {
  # Sample population parameters from hyperpriors
  mu_b0 <- rnorm(1, 0, 5)
  sigma_b0 <- abs(rt(1, df = 2))
  mu_b1 <- rnorm(1, -3, 2)
  sigma_b1 <- abs(rt(1, df = 2))
  mu_omega <- rnorm(1, -5, 5)
  sigma_omega <- abs(rt(1, df = 2))
  mu_sm <- rnorm(1, 0, 2)
  sigma_sm <- abs(rt(1, df = 2))
  eta <- rlnorm(1, 2, 1)
  # Sample individual parameters
  b0 <- rnorm(1, mu_b0, sigma_b0)
```

```
  b1 <- rlnorm(1, mu_b1, sigma_b1)
  omega <- rlnorm(1, mu_omega, sigma_omega)
  sigma_sensory <- rlnorm(1, mu_sm, sigma_sm)
  # Initialize
  var_psi <- 10000
  mu_psi <- U / (1 + exp(-(b0 + b1 * stim_values[1])))
  responses <- numeric(n_trials)
  variances <- numeric(n_trials)
  # Simulate trial sequence
  for (j in 1:n_trials) {
    stim <- sample(stim_values, 1)
    mu_sensory <- U / (1 + exp(-(b0 + b1 * stim)))
    # Kalman update
    K <- var_psi / (var_psi + sigma_sensory^2)
    mu_psi <- mu_psi + K * (mu_sensory - mu_psi)
    var_psi <- (1 - K) * var_psi + eta * exp(-omega * j)
    responses[j] <- rnorm(1, mu_psi, sqrt(var_psi))
    variances[j] <- var_psi
  }
  # Evaluate
  results <- rbind(results, data.frame(
    sim = s,
    prop_in_range = mean(responses >= 0 & responses <= U),
    mean_response = mean(responses),
    var_decreases = variances[n_trials] < variances[1]
  ))
}
```

iv. Conduct prior sensitivity analysis (after model fitting):

Note: This checks whether conclusions depend on specific prior choices. Refit the model with alternative reasonable priors and compare the perceptual distribution estimates. High correlations between estimates from different priors indicate robustness; low correlations warrant caution in interpretation.

10. Create and fit the JAGS model.
    a. Create the JAGS model specification file:

```
cat('
model {
```

```
for (i in 1:Nparticipants) {
  for (j in 1:Ntrials) {
    # Response likelihood
    psi[i,j] ~ dnorm(mu_psi[i,j], 1/var_psi[i,j])
    # Posterior predictive
    psi_pred[i,j] ~ dnorm(mu_psi[i,j], 1/var_psi[i,j])
    # Kalman gain (variance-based)
    K[i,j+1] <- var_psi[i,j] / (var_psi[i,j] + var_sensory[i])
  # Variance update with decaying process noise
  var_psi[i,j+1] <- (1 - K[i,j+1]) * var_psi[i,j] +
        eta * exp(-omega[i] * j)
  # Mean update
  mu_psi[i,j+1] <- mu_psi[i,j] + K[i,j+1] *
         (mu_sensory[i,stim[i,j+1]] - mu_psi[i,j])
  }
  # Initialization
  mu_psi[i,1] <- mu_sensory[i,stim[i,1]]
  var_psi[i,1] <- 10000
  # Sensory mapping for each stimulus
  for (k in 1:Nstimuli) {
  mu_sensory[i,k] <- U / (1 + exp(-(b0[i] + b1[i] * size[k])))
  }
  # Sensory variance from SD parameter
  var_sensory[i] <- pow(sigma_sensory[i], 2)
  # Individual parameters from population distributions
  b0[i] ~ dnorm(mu_b0, tau_b0)
  b1[i] ~ dlnorm(mu_b1, tau_b1)
  omega[i] ~ dlnorm(mu_omega, tau_omega)
  sigma_sensory[i] ~ dlnorm(mu_sm, tau_sm)
}
# Population-level hyperpriors
mu_b0 ~ dnorm(0, 1/25)
tau_b0 ~ dscaled.gamma(2, 1)
sigma_b0 <- 1/sqrt(tau_b0)
mu_b1 ~ dnorm(-3, 1/4)
tau_b1 ~ dscaled.gamma(2, 1)
sigma_b1 <- 1/sqrt(tau_b1)
```

```
  mu_omega ~ dnorm(-5, 1/25)
  tau_omega ~ dscaled.gamma(2, 1)
  sigma_omega <- 1/sqrt(tau_omega)
  mu_sm ~ dnorm(0, 1/4)
  tau_sm ~ dscaled.gamma(2, 1)
  sigma_sm <- 1/sqrt(tau_sm)
  eta ~ dlnorm(2, 1)
  # Constant
  U <- 200
}
', file = "models/jags_models/perceptual_model.txt")
```

b. Fit the model:

```
# Prepare data
jags_data <- list(
  psi = perceptual_responses,
  stim = stimulus_indices,
  size = stimulus_values,
  Nparticipants = n_participants,
  Ntrials = n_trials,
  Nstimuli = n_stimuli
)
# Parameters to monitor
params <- c("mu_psi", "var_psi", "K", "psi_pred",
            "b0", "b1", "omega", "sigma_sensory",
            "mu_b0", "sigma_b0", "mu_b1", "sigma_b1",
            "mu_omega", "sigma_omega", "eta")
# Fit model
fit <- jags(
  data = jags_data,
  parameters.to.save = params,
  model.file = "models/jags_models/perceptual_model.txt",
  n.chains = 4,
  n.iter = 100000,
  n.burnin = 75000,
```

```
  n.thin = 10,
  parallel = TRUE
)
```

c. Check convergence:

```
# All R-hat values should be < 1.05
rhat_vals <- unlist(fit$Rhat)
max_rhat <- max(rhat_vals, na.rm = TRUE)
if (max_rhat > 1.05) {
  warning("Convergence issues. Consider more iterations.")
}
# Visual check for key hyperparameters
mcmc_trace(fit$samples,
     pars = c("mu_b0", "mu_b1", "mu_omega", "eta"))
```

11. Validate the model.
    a. Conduct posterior predictive checks:

```
# Compute observed and predicted statistics by stimulus
ppc_data <- data.frame()
for (k in 1:n_stimuli) {
  # Find all observations of stimulus k
  mask <- which(stimulus_indices == k, arr.ind = TRUE)
  obs_vals <- perceptual_responses[mask]
  pred_vals <- fit$mean$psi_pred[mask]
  ppc_data <- rbind(ppc_data, data.frame(
    stimulus = stimulus_values[k],
    obs_mean = mean(obs_vals, na.rm = TRUE),
    obs_sd = sd(obs_vals, na.rm = TRUE),
    pred_mean = mean(pred_vals, na.rm = TRUE),
    pred_sd = sd(pred_vals, na.rm = TRUE)
  ))
}
# Plot means
p1 <- ggplot(ppc_data, aes(x = stimulus)) +
  geom_point(aes(y = obs_mean), size = 3) +
```

```
  geom_line(aes(y = pred_mean), color = "steelblue", linewidth = 1) +
  geom_point(aes(y = pred_mean), color = "steelblue", size = 2) +
  labs(x = "Physical Stimulus (mm)", y = "Perceptual Response",
    title = "Posterior Predictive: Means") +
  theme_minimal()
# Plot SDs
p2 <- ggplot(ppc_data, aes(x = stimulus)) +
  geom_point(aes(y = obs_sd), size = 3) +
  geom_line(aes(y = pred_sd), color = "steelblue", linewidth = 1) +
  geom_point(aes(y = pred_sd), color = "steelblue", size = 2) +
  labs(x = "Physical Stimulus (mm)", y = "Response SD",
    title = "Posterior Predictive: Variability") +
  theme_minimal()
print(p1 + p2)
ggsave("outputs/figures/perceptual_ppc.pdf", width = 12, height = 5)
```

Note: Good model fit is indicated when predicted values track observed patterns across stimuli for both means and variability.

b. Check perceptual dynamics:

```
# Mean Kalman gain trajectory across participants
K_mean <- colMeans(fit$q50$K[, 2:(n_trials+1)], na.rm = TRUE)
plot(1:n_trials, K_mean, type = "l", ylim = c(0, 1),
  xlab = "Trial", ylab = "Kalman Gain (K)",
  main = "Perceptual Dynamics: Weighting Over Trials")
abline(h = 0.5, lty = 2, col = "gray")
# Also check variance dynamics
var_mean <- colMeans(fit$q50$var_psi, na.rm = TRUE)
plot(1:n_trials, var_mean, type = "l",
  xlab = "Trial", ylab = expression(sigma[psi]^2),
  main = "Perceptual Dynamics: Uncertainty Over Trials")
```

c. Interpret expected patterns:

Note: K should start high and decrease, reflecting increasing reliance on accumulated experience. σ_ψ^2 should decrease over trials, reflecting increasing perceptual precision. If K remains near 1 throughout, sensory variance may be underestimated. If σ_ψ^2 does not decrease, process noise (η) may be too large.

12. Extract results for distance computation.
    a. Save the outputs for use in Stimulus distance computation:

```
# Extract posterior medians of perceptual distributions

mu_psi <- fit$q50$mu_psi  # [Nparticipants x Ntrials] means

var_psi <- fit$q50$var_psi  # [Nparticipants x Ntrials] variances

sigma_psi <- sqrt(var_psi)  # Convert to SD for distance computation

# Save for next step

save(mu_psi, sigma_psi, var_psi, fit,

  file = "models/fitted_models/perceptual_results.RData")
```

    b. Understand the outputs.

Note: The outputs $\mu_{\psi,ij}$ and $\sigma_{\psi,ij}$ define trial-by-trial perceptual distributions $\mathcal{N}\left(\mu_{\psi,ij}, \sigma^2_{\psi,ij}\right)$ for each participant. These distributions characterize both the central tendency and uncertainty of perception at each moment, forming the basis for computing distribution-based perceptual distances in the next step.

Stimulus distance computation

Timing: 1–2 h

Note: Compute probabilistic perceptual distance measures from the Bayesian perceptual model results to create distribution-based perceptual distance inputs for the hierarchical mixture generalization model. This step bridges the perceptual data stream and generalization data stream by transforming probabilistic perceptual representations into similarity metrics. Three types of stimulus distance measures enable comparison of different perceptual assumptions: physical distances assume veridical perception, point-based perceptual distances use observed ratings directly, and distribution-based perceptual distances account for perceptual uncertainty through distribution overlap calculations.

Note: In Yu et al. (2025a),[2] CS memory is assumed to be the same as the most recent CS encoding. However, this framework can be extended by measuring CS memory explicitly in experiments and modeling CS memory distributions, as demonstrated in Yu et al. (2024).[19]

13. Load Bayesian perceptual model results and initialize distance matrices.
    a. Set up the computational workspace:

```
# Load fitted Bayesian perceptual model results

load("models/fitted_models/perceptual_model_results.RData")

# Extract key perceptual estimates (posterior medians)

mu_psi <- perceptual_model$q50$mu_psi # Perceptual means
```

```
sigma_psi <- perceptual_model$q50$sigma_psi # Perceptual SDs

# Get experimental dimensions

n_participants <- dim(mu_psi)[1]

n_trials <- dim(mu_psi)[2]

# Initialize distance matrices for three approaches

physical_dist <- array(NA, c(n_participants, n_trials))

point_based_perceptual_dist <- array(NA,

                 c(n_participants, n_trials))

distribution_based_perceptual_dist <- array(NA,

                 c(n_participants, n_trials))
```

14. Compute physical distances (veridical perception assumption).
    a. Calculate distances based on objective stimulus properties:

```
# Load experimental design parameters

load("data/processed/experimental_design.RData")

# Calculate physical distances for each participant

for (i in 1:n_participants) {

  # Identify CS value for this participant

  cs_value <- stimulus_values[cs_assignment[i]]

  # Calculate absolute differences from CS

  for (j in 1:n_trials) {

    current_stimulus <- stimulus_values[stimulus_sequence[i, j]]

    physical_dist[i, j] <- abs(current_stimulus - cs_value)

  }

}

cat("Physical distances computed.\n")
```

15. Compute point-based perceptual distances (observed ratings approach).
    a. Calculate distances using participant-specific CS representations:

```
# Load original perceptual response data

load("data/processed/perceptual_responses.RData")

# Calculate point-based perceptual distances

for (i in 1:n_participants) {

  # Identify CS trials for this participant

  cs_trials <- which(trial_type[i, ] == "CS")

  if (length(cs_trials) > 0) {

    # Calculate mean CS perception from observed ratings

    cs_mean_perception <- mean(perceptual_responses[i, cs_trials],
```

```
                  na.rm = TRUE)
  # Calculate absolute differences from CS mean
  point_based_perceptual_dist[i, ] <-
    abs(perceptual_responses[i, ] - cs_mean_perception)
  }
}
```

16. Build dynamic CS memory representations for distribution-based approach.
    a. Implement evolving CS memory based on encounter history:

```
# Initialize CS memory tracking matrices
cs_memory_mean <- array(NA, c(n_participants, n_trials))
cs_memory_sd <- array(NA, c(n_participants, n_trials))
# Build CS memory representations trial-by-trial
for (i in 1:n_participants) {
  # Initialize first trial with initial perception
  cs_memory_mean[i, 1] <- mu_psi[i, 1]
  cs_memory_sd[i, 1] <- sigma_psi[i, 1]
  # Update memory based on CS encounter history
  for (j in 2:n_trials) {
    if (trial_type[i, j - 1] == "CS") {
      # Update memory with most recent CS perception
      cs_memory_mean[i, j] <- mu_psi[i, j - 1]
      cs_memory_sd[i, j] <- sigma_psi[i, j - 1]
    } else {
      # Maintain previous CS memory
      cs_memory_mean[i, j] <- cs_memory_mean[i, j - 1]
      cs_memory_sd[i, j] <- cs_memory_sd[i, j - 1]
    }
  }
}
```

17. Compute distribution-based perceptual distances via Monte Carlo simulation (see Figure 2 as example).
    a. Apply distribution overlap-based distance formula:

$$\text{distance}_{ij} = 1 - \int \min\left(f_{\text{current}}(x), f_{\text{memory}}(x)\right) dx \qquad \text{(Equation 17)}$$

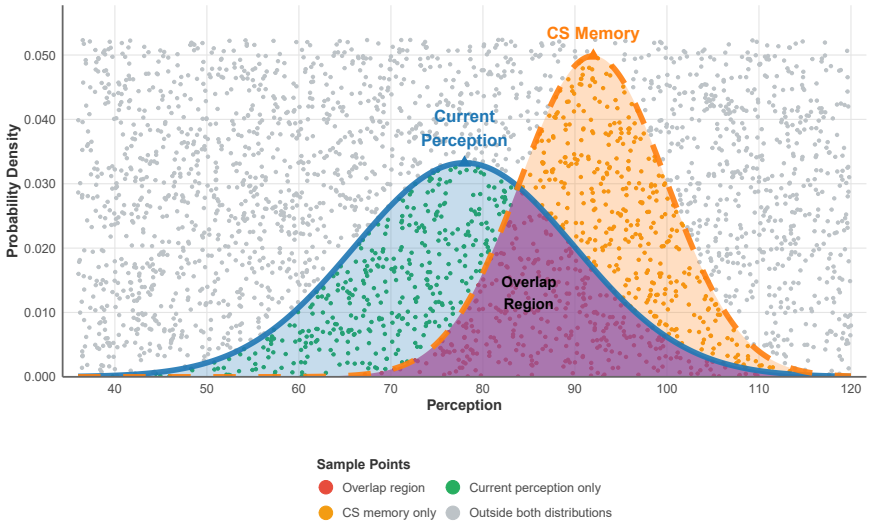


Figure 2. Monte Carlo estimation of perceptual distribution overlap

Two normal distributions represent current perception (blue, μ=78, σ=12) and CS memory (orange, μ=92, σ=8). Randomly sampled points (n=3,000) are colored by their classification: overlap region (red), current perception only (green), CS memory only (orange), and outside both distributions (gray). The purple shaded area shows the overlap region between the two probability density functions. This Monte Carlo approach estimates the overlap coefficient as the average proportion of overlap relative to each distribution, yielding a perceptual distance measure of 1 − overlap coefficient $\left(\frac{\#red}{\#green+\#red+\#orange}\right)$.

b. Define distribution functions:

$$f_{current}(x) = \mathcal{N}\left(\mu_{\psi.ij}, \sigma^2_{\psi.ij}\right) \quad \text{(Equation 18)}$$

$$f_{memory}(x) = \mathcal{N}\left(\mu^{CS}_{ij}, \left(\sigma^{CS}_{ij}\right)^2\right) \quad \text{(Equation 19)}$$

c. Use Monte Carlo simulation (Figure 2) to approximate the integral:

Note: The integral represents the overlapping area between two distributions. Monte Carlo approximation randomly scatters points across the region. Each point is classified as falling within the current perception area, CS memory area, overlap region, or neither. The proportion of points in each region estimates the area under curves.

d. Implement Monte Carlo integral approximation:

```
# Function to approximate the overlap integral via Monte Carlo

overlap_integral <- function(mu1, sd1, mu2, sd2) {

  # Set seed for reproducibility

  set.seed(2025)

  n_samples <- 100000

  # Define integration bounds: +/-3 SD from both distributions

  x_min <- min(mu1 - 3 * sd1, mu2 - 3 * sd2)

  x_max <- max(mu1 + 3 * sd1, mu2 + 3 * sd2)
```

```
  # Define y-axis bounds for 2D sampling space
  x_grid <- seq(x_min, x_max, length.out = n_samples)
  f1_vals <- dnorm(x_grid, mean = mu1, sd = sd1)
  f2_vals <- dnorm(x_grid, mean = mu2, sd = sd2)
  y_max <- max(f1_vals, f2_vals)
  # Generate random points for Monte Carlo area estimation
  sample_points <- matrix(
    c(runif(n_samples, x_min, x_max),
      runif(n_samples, 0, y_max)),
    ncol = 2
  )
  # Count points falling under each distribution curve
  under_current <- sample_points[, 2] <=
    dnorm(sample_points[, 1], mu1, sd1)
  under_memory <- sample_points[, 2] <=
    dnorm(sample_points[, 1], mu2, sd2)
  # Count points in overlap region: min(f_current, f_memory)
  under_overlap <- sample_points[, 2] <=
    pmin(dnorm(sample_points[, 1], mu1, sd1),
      dnorm(sample_points[, 1], mu2, sd2))
  # Approximate integral using relative overlap proportions
  if (sum(under_current) > 0 && sum(under_memory) > 0) {
    integral_approx <-
      (sum(under_overlap) / sum(under_current) +
        sum(under_overlap) / sum(under_memory)) / 2
  } else {
    integral_approx <- 0 # No overlap possible
  }
  return(integral_approx)
}
# Calculate distribution-based perceptual distances
cat("Computing distribution-based perceptual distances",
  "via Monte Carlo integration...\n")
  for (i in 1:n_participants) {
  for (j in 1:n_trials) {
    # Check for missing values
    required_values <- c(mu_psi[i, j], sigma_psi[i, j],
              cs_memory_mean[i, j], cs_memory_sd[i, j])
```

```
    if (any(is.na(required_values))) {
      distribution_based_perceptual_dist[i, j] <- NA
    next
  }
    # Apply distance formula: 1 - integral
    overlap_integral_value <- overlap_integral(
      mu_psi[i, j], sigma_psi[i, j],
      cs_memory_mean[i, j], cs_memory_sd[i, j]
    )
    distribution_based_perceptual_dist[i, j] <-
      1 - overlap_integral_value
  }
# Progress indicator
if (i %% 10 == 0) {
cat("Completed participant", i, "of", n_participants, "\n")
  }
}
```

18. Save distance matrices for generalization modeling.
    a. Prepare and export distance measures:

```
# Rename variables for consistency with generalization model
d_phy <- physical_dist
d_per_old <- point_based_perceptual_dist
d_per_new <- distribution_based_perceptual_dist
# Save all distance matrices
save(d_phy, d_per_old, d_per_new, cs_memory_mean, cs_memory_sd,
  file = "data/processed/distance_matrices.RData")
```

Note: These distance matrices serve as inputs for the hierarchical mixture generalization model, enabling comparison between point-based and distribution-based perceptual assumptions.

Hierarchical mixture generalization model

Timing: 10–24 h

Note: Model fear generalization using a hierarchical mixture approach that processes the generalization data stream from data processing and incorporates the distance measures from stimulus distance computation. The model identifies distinct generalization pathways across participants, integrating Rescorla-Wagner learning[21] with similarity-based

generalization.[22] We fit two identical versions differing only in perceptual distance assumptions: point-based (d_per_old)[1] versus distribution-based (d_per_new).[2]

19. Understand the computational framework and latent group structure.

Note: The model identifies four theoretically-motivated latent pathways (see Yu et al. (2023)[1]): (1) **Non-Learners**, characterized by no fear acquisition (α=0) and flat responses; (2) **Overgeneralizers**, who learn CS-US associations but generalize broadly, maintaining >70% fear response across all stimuli; (3) **Physical Generalizers**, showing distance-dependent generalization based on objective physical stimulus similarity; (4) **Perceptual Generalizers**, showing distance-dependent generalization based on individual perceptual representations.

Note: The core computational equations are:

$$V_{ij+1} = V_{ij} + \alpha_i\left(R_{ij} - V_{ij}\right)\cdot CS_{ij} \quad \text{(Learning)} \qquad \text{(Equation 20)}$$

$$S_{ij} = \begin{cases} 1 & \text{if Non} - \text{Learner or Overgeneralizer} \\ \exp\left(-\lambda_i \cdot D_{ij}^{\text{phy}}\right) & \text{if Physical Generalizer} \\ \exp\left(-\lambda_i \cdot D_{ij}^{\text{per}}\right) & \text{if Perceptual Generalizer} \end{cases} \qquad \text{(Equation 21)}$$

$$G_{ij} = V_{ij}\cdot S_{ij} \quad \text{(Generalized strength)} \qquad \text{(Equation 22)}$$

$$\theta_{ij} = A + \frac{K - A}{1+\exp\left(-\left(w_{0i}+w_{1i}\cdot G_{ij}\right)\right)} \quad \text{(Response)} \qquad \text{(Equation 23)}$$

where V = learned strength, S = similarity, G = generalized strength, θ = expected response, A=1, K=10 for 1-10 rating scales.

a. Calculate the overgeneralization threshold constraint:

```
# Compute threshold ensuring >70% generalization at max distance

max_dist <- max(c(d_phy, d_per_old, d_per_new), na.rm = TRUE)

overgen_threshold <- -log(0.7) / max_dist
```

Note: The constraint $exp(-\lambda \cdot d_{max})>0.7$ yields $\lambda < \frac{-\log(0.7)}{d_{max}}$.

20. Review parameter summary.
    a. Reference the parameter table:

Parameter	Interpretation	Range	Level	Groups
α_i	Learning rate	[0,1]	Individual	2,3,4
λ_i	Generalization rate	>0	Individual	3,4
$w_{0,i}$	Response baseline	$\mathbb{R}$	Individual	All
$w_{1,i}$	Response scaling	>1	Individual	2,3,4
σ	Response noise	>0	Group	All
π	Group probabilities	Simplex	Population	All

b. Review group-specific parameter constraints.

Note: Non-Learners (Group 1) have α=0 and w_1=0 by construction. Overgeneralizers (Group 2) have λ constrained below threshold. w_1>1 for Groups 2–4 ensures identifiability; values near 0 would produce nearly flat responses indistinguishable from Non-Learners.

c. Understand the beta reparameterization for learning rates:

$$\alpha_i \sim \text{Beta}(\alpha_a, \alpha_b), \alpha_a = \alpha_\mu \cdot \alpha_\kappa, \alpha_b = (1 - \alpha_\mu) \cdot \alpha_\kappa \quad \text{(Equation 24)}$$

where $\alpha_\mu \in (0,1)$ is the population mean and α_κ>0 controls concentration.

21. Specify priors and validate them.

a. Specify hierarchical structure for individual parameters:

$$\alpha_i \sim \text{Beta}(\alpha_\mu \cdot \alpha_\kappa, (1 - \alpha_\mu) \cdot \alpha_\kappa) \quad \text{(Equation 25)}$$

$$\lambda_i \sim \mathcal{N}^+(\mu_\lambda, \sigma_\lambda^2) \text{ with group } - \text{ specific truncation} \quad \text{(Equation 26)}$$

$$w_{0,i} \sim \mathcal{N}(\mu_{w_0}, \sigma_{w_0}^2) \quad \text{(Equation 27)}$$

$$w_{1,i} \sim \text{Gamma}(a_{w_1}, b_{w_1}) \cdot \mathbb{I}(w_{1,i} > 1) \quad \text{(Equation 28)}$$

b. Specify population-level hyperpriors:

$$\alpha_\mu \sim \text{Beta}(1,1) \alpha_\kappa \sim \text{Uniform}(1, 10) \quad \text{(Equation 29)}$$

$$\mu_\lambda \sim \text{LogNormal}(0, 0.5^2) \sigma_\lambda \sim \text{Uniform}(0, 1) \quad \text{(Equation 30)}$$

$$\mu_{w_0} \sim \mathcal{N}(0, 10^2) \sigma_{w_0} \sim \text{half} - t(2) \quad \text{(Equation 31)}$$

$$a_{w_1}, b_{w_1} \sim \text{half} - t(2) \quad \text{(Equation 32)}$$

c. Specify group-level priors:

$$\pi \sim \text{Dirichlet}(1,1,1,1) \quad \text{(Equation 33)}$$

$$\sigma_{\text{learners}} \sim \text{Uniform}(0, 1.5) \quad \text{(Equation 34)}$$

$$\sigma_{\text{non-learners}} \sim \text{Uniform}(1.5, 3) \quad \text{(Equation 35)}$$

Note: Separate noise priors reflect that Non-Learners produce random responses (higher noise) while learners show systematic patterns. These priors were validated in Yu et al. (2025)[2] with prior sensitivity check.

d. Conduct prior predictive checking (recommended for new contexts):

```
set.seed(123)

n_sim <- 1000

A <- 1; K <- 10 # Response scale

# Sample hyperparameters

  alpha_mu_sim <- rbeta(n_sim, 1, 1)
```

```
  alpha_kappa_sim <- runif(n_sim, 1, 10)
lambda_mu_sim <- rlnorm(n_sim, 0, 0.5)
w0_mu_sim <- rnorm(n_sim, 0, 10)
w1_a_sim <- abs(rt(n_sim, 2)) + 0.01
w1_b_sim <- abs(rt(n_sim, 2)) + 0.01
# Sample individual parameters
alpha_sim <- rbeta(n_sim,
alpha_mu_sim * alpha_kappa_sim,
(1 - alpha_mu_sim) * alpha_kappa_sim)
lambda_sim <- rlnorm(n_sim, log(lambda_mu_sim), 0.5)
w0_sim <- rnorm(n_sim, w0_mu_sim, 2)
w1_sim <- rgamma(n_sim, w1_a_sim, w1_b_sim)
w1_sim <- pmax(w1_sim, 1) # Truncate at 1
# Simulate responses across range of G values
G_values <- seq(0, 1, 0.1)
theta_sim <- matrix(NA, n_sim, length(G_values))
for (g in seq_along(G_values)) {
  theta_sim[, g] <- A + (K - A) /
    (1 + exp(-(w0_sim + w1_sim * G_values[g])))
}
```

22. Prepare data for model fitting.
    a. Load data and create model inputs:

```
# Set seed for reproducibility
set.seed(2025)
# Load distance matrices
load("data/processed/distance_matrices.RData")
max_dist <- max(c(d_phy, d_per_old, d_per_new), na.rm = TRUE)
overgen_threshold <- -log(0.7) / max_dist
# Point-based model data
data_point <- list(
  y = expectancy_responses, d_phy = d_phy,
  d_per_old = d_per_old,
  k = cs_indicators, r = reinforcement_outcomes,
  Nparticipants = n_participants, Ntrials = n_trials,
  overgen_limit = overgen_threshold
)
# Distribution-based model data
```

```
data_dist <- list(
  y = expectancy_responses, d_phy = d_phy,
  d_per_new = d_per_new,
  k = cs_indicators, r = reinforcement_outcomes,
  Nparticipants = n_participants, Ntrials = n_trials,
  overgen_limit = overgen_threshold
)
```

23. Create JAGS model specification.
    a. Write the generalization model:

```
cat('
model{
  for (i in 1:Nparticipants) {
    for (j in 1:Ntrials) {
    # Response likelihood with group-specific noise
    y[i,j] ~ dnorm(theta[i,j], 1/sigma[zn[i]]^2)
    # Posterior predictive samples
    y_pred[i,j] ~ dnorm(theta[i,j], 1/sigma[zn[i]]^2)
    # Sigmoid transformation
    theta[i,j] <- A + (K-A) / (1 + exp(-(w0[i] + w1[i] * g[i,j])))
    # Generalization: learned strength x similarity
    g[i,j] <- v[i,j] * s[i,j]
    # Group-dependent similarity
    s[i,j] <- ifelse(v[i,j] > 0 && gp[i] > 1,
         exp(-lambda[i] * ifelse(gp[i] == 3,
           d_phy[i,j], d_per_old[i,j])), 1)
    # Rescorla-Wagner learning
    v[i,j+1] <- ifelse(gp[i] != 1,
         ifelse(k[i,j] == 1,
           v[i,j] + alpha[i] * (r[i,j] - v[i,j]),
           v[i,j]), 0)
    }
    v[i,1] <- 0
    # Group assignment
    gp[i] ~ dcat(pi[])
    # Lambda with group-specific constraints
    lambda_1[i] ~ dnorm(lambda_mu, 1/lambda_sigma^2)T(overgen_limit,)
    lambda_2[i] ~ dnorm(lambda_mu, 1/lambda_sigma^2)T(0, overgen_limit)
```

```
    lambda[i] <- ifelse(gp[i] == 1, 0,
           ifelse(gp[i] == 2, lambda_2[i], lambda_1[i]))
    # Response parameters
    w0[i] ~ dnorm(w0_mu, w0_sigma)
    # T(1,): ensures identifiability from Non-Learners
    w1_1[i] ~ dgamma(w1_a, w1_b)T(1,)
    w1[i] <- ifelse(gp[i] == 1, 0, w1_1[i])
    # Learning rate
    alpha_1[i] ~ dbeta(alpha_a, alpha_b)T(1e-9, 1-1e-9)
    alpha[i] <- ifelse(gp[i] == 1, 0, alpha_1[i])
    zn[i] <- ifelse(gp[i] == 1, 2, 1)
  }
  # Hyperpriors
  lambda_mu ~ dlnorm(0, pow(.5, -2))
  lambda_sigma ~ dunif(1e-9, 1)
  alpha_a <- alpha_mu * alpha_kappa
  alpha_b <- (1 - alpha_mu) * alpha_kappa
  alpha_mu ~ dbeta(1, 1)T(1e-9, 1-1e-9)
  alpha_kappa ~ dunif(1, 10)
  sigma[1] ~ dunif(1e-9, 1.5)
  sigma[2] ~ dunif(1.5, 3)
  pi[1:4] ~ ddirch(c(1, 1, 1, 1))
  w1_a ~ dscaled.gamma(2, 1)T(1e-9,)
  w1_b ~ dscaled.gamma(2, 1)T(1e-9,)
  w0_mu ~ dnorm(0, 1/10^2)
  w0_sigma ~ dscaled.gamma(2, 1)
  A <- 1
  K <- 10
}', file = "models/jags_models/gen_point.txt")
```

b. Create distribution-based model by substitution:

```
# Only difference: d_per_old -> d_per_new
point_code <- readLines("models/jags_models/gen_point.txt")
dist_code <- gsub("d_per_old", "d_per_new", point_code)
writeLines(dist_code, "models/jags_models/gen_dist.txt")
```

24. Fit both model versions using MCMC.
    a. Configure and run MCMC:

```
params <- c("gp", "pi", "alpha", "lambda", "w0", "w1", "sigma",
       "alpha_mu", "lambda_mu", "lambda_sigma", "y_pred")
mcmc_set <- list(n.chains = 4, n.iter = 150000,
       n.burnin = 100000, n.thin = 25)
# Fit point-based model
model_point <- do.call(jags, c(
  list(data = data_point,
    model.file = "models/jags_models/gen_point.txt",
    parameters.to.save = params), mcmc_set))
# Fit distribution-based model
model_dist <- do.call(jags, c(
  list(data = data_dist,
    model.file = "models/jags_models/gen_dist.txt",
    parameters.to.save = params), mcmc_set))
```

b. Check convergence:

```
# Verify convergence
max_rhat <- max(c(model_point$Rhat, model_dist$Rhat), na.rm = TRUE)
if (max_rhat > 1.1) {
  warning("Convergence issues detected (R-hat > 1.1).")
}
# Visual diagnostics
mcmc_trace(model_point$samples, pars = c("alpha_mu", "lambda_mu"))
```

25. Perform posterior predictive checks.
    a. Extract and organize predictions:

```
n_pred <- 5000
idx <- sample(1:nrow(model_point$sims.list$y_pred), n_pred)
y_pred_point <- model_point$sims.list$y_pred[idx, , ]
y_pred_dist <- model_dist$sims.list$y_pred[idx, , ]
y_obs <- data_point$y
```

b. Compute stimulus-wise statistics:

```
stim_labels <- rep(1:n_stimuli, each = n_participants)
obs_stats <- data.frame(stimulus = stim_labels,
          y = as.vector(y_obs)) %>%
  group_by(stimulus) %>%
```

```
  summarise(mean_obs = mean(y, na.rm = TRUE),
            q10_obs = quantile(y, 0.1, na.rm = TRUE),
            q50_obs = quantile(y, 0.5, na.rm = TRUE),
            q90_obs = quantile(y, 0.9, na.rm = TRUE))
compute_pred <- function(y_pred) {
  data.frame(stimulus = rep(stim_labels, n_pred),
             y = as.vector(y_pred)) %>%
  group_by(stimulus) %>%
  summarise(mean_pred = mean(y, na.rm = TRUE),
            q10_pred = quantile(y, 0.1, na.rm = TRUE),
            q50_pred = quantile(y, 0.5, na.rm = TRUE),
            q90_pred = quantile(y, 0.9, na.rm = TRUE))
}
ppc_point <- merge(obs_stats, compute_pred(y_pred_point))
ppc_dist <- merge(obs_stats, compute_pred(y_pred_dist))
```

c. Create and save plots:

```
# Plot observed vs predicted means
ggplot(ppc_point, aes(x = stimulus)) +
  geom_point(aes(y = mean_obs), size = 3) +
  geom_line(aes(y = mean_pred), color = "steelblue", linewidth = 1) +
  labs(x = "Stimulus", y = "Fear Response",
    title = "PPC: Point-based Model") +
  theme_minimal()
ggsave("outputs/figures/ppc_gen_point.pdf", width = 8, height = 5)
ggplot(ppc_dist, aes(x = stimulus)) +
  geom_point(aes(y = mean_obs), size = 3) +
  geom_line(aes(y = mean_pred), color = "firebrick", linewidth = 1) +
  labs(x = "Stimulus", y = "Fear Response",
    title = "PPC: Distribution-based Model") +
  theme_minimal()
ggsave("outputs/figures/ppc_gen_dist.pdf", width = 8, height = 5)
```

26. Extract and compare model results.

a. Analyze group allocation patterns:

```
get_modal_group <- function(gp_samples) {
  apply(gp_samples, 2, function(x) {
    as.numeric(names(sort(table(x), decreasing = TRUE))[1])
```

```
  })
}
groups_point <- get_modal_group(model_point$sims.list$gp)
groups_dist <- get_modal_group(model_dist$sims.list$gp)
group_labels <- c("Non-Learner", "Overgeneralizer",
         "Physical Generalizer", "Perceptual Generalizer")
# Display allocations
table(factor(groups_point, levels = 1:4, labels = group_labels))
table(factor(groups_dist, levels = 1:4, labels = group_labels))
```

b. Compare group probabilities and save:

```
comparison <- data.frame(
  Group = group_labels,
  Point = round(model_point$mean$pi, 3),
  Dist = round(model_dist$mean$pi, 3),
  Diff = round(model_dist$mean$pi - model_point$mean$pi, 3)
)
print(comparison)
# Save results
save(model_point, model_dist, groups_point, groups_dist,
    comparison, ppc_point, ppc_dist, overgen_threshold,
    file = "models/fitted_models/generalization_results.RData")
```

Note: Use these models for model comparison framework to evaluate point-based versus distribution-based perceptual assumptions through formal Bayes factor computation.

Model comparison framework

Timing: 12–24 h

Note: Compare distribution-based versus point-based perceptual assumptions using a nested super-model approach and Bayes factor computation. This framework combines both hierarchical mixture generalization models into a unified comparison structure. The super-model concept combines two competing models into a single nested structure using a model selection parameter (β_M) that continuously weights between them. Rather than fitting separate models and comparing them post-hoc, this approach estimates the relative evidence for each formal model simultaneously within the same inferential framework. The key advantage is that it provides formal Bayes factor computation, avoiding approximation errors inherent in information criteria comparisons. This method directly quantifies how much the data favor one theoretical assumption over another, making the model comparison statistically principled and interpretable.

27. Understand the super-model theoretical framework.

Note: The nested model structure combines both perceptual approaches:

$$L_{\text{combined}} = \beta_M \cdot L_{\text{distribution\_based}} + (1 - \beta_M) \cdot L_{\text{point\_based}} \quad \text{(Equation 36)}$$

$$\beta_M \sim \text{Beta}(1, 1) \quad (\text{Model selection parameter}) \quad \text{(Equation 37)}$$

where each model has independent parameter sets and group assignments.

Note: JAGS requires standard probability distributions for likelihoods. The zero trick implements custom likelihoods: $0 \sim \text{Poisson}(-log(L_{\text{custom}}) + C)$. Each participant can have different group memberships under different perceptual assumptions. The model selection parameter β_M determines which approach better explains the generalization data.

28. Load required data and prepare super-model inputs.
    a. Load distance matrices and experimental data:

```
# Load distance matrices from stimulus distance computation
load("data/processed/distance_matrices.RData")
load("data/processed/experimental_data.RData")
# Prepare super-model data list
super_data <- list(
  y = expectancy_responses,
  d_phy = d_phy,
  d_per_old = d_per_old, # Point-based perceptual distances
  d_per_new = d_per_new, # Distribution-based perceptual distances
  k = cs_indicators,
  r = reinforcement_outcomes,
  Nparticipants = n_participants,
  Ntrials = n_trials,
  overgen_limit = overgen_threshold, # Overgeneralization threshold
  zero = rep(0, n_participants)
)
```

29. Implement super-model JAGS specification.
    a. Create the super-model file matching your original structure:

```
# Create super-model specification
cat('
data{
  for (i in 1:Nparticipants){
    zero[i] = 0
  }
  C = 1000000
}
```

```
model{
  for (i in 1:Nparticipants) {
    # Zero trick combining likelihoods from both complete models
    zero[i] ~ dpois(-log(pQ * prod(lh1[i,]) +
             (1-pQ) * prod(lh2[i,])) + C)
    for (j in 1:Ntrials) {
      # Model 1: Distribution-based perceptual distances
      lh1[i,j] = dnorm(y[i,j], theta[i,j,1],
             1/sigma[zn[i,1],1] ^2)
      # Model 2: Point-based perceptual distances
      lh2[i,j] = dnorm(y[i,j], theta[i,j,2],
             1/sigma[zn[i,2],2] ^2)
    # Placeholder distribution (required by JAGS)
    y[i,j] ~ dunif(1, 10)
    # Response transformation for both models
    theta[i,j,1] <- 1 + (10-1) /
      (1 + exp(-(w0[i,1] + w1[i,1] * g[i,j,1])))
    theta[i,j,2] <- 1 + (10-1) /
      (1 + exp(-(w0[i,2] + w1[i,2] * g[i,j,2])))
    # Generalization for Model 1 (distribution-based perceptual distances)
    g[i,j,1] <- v[i,j,1] * s[i,j,1]
    s[i,j,1] <- ifelse(v[i,j,1] > 0 && gp[i,1] > 1,
           exp(-lambda[i] *
             ifelse(gp[i,1] == 3,
                d_phy[i,j],
                d_per_new[i,j])), 1)
    # Generalization for Model 2 (point-based perceptual distances)
    g[i,j,2] <- v[i,j,2] * s[i,j,2]
    s[i,j,2] <- ifelse(v[i,j,2] > 0 && gp[i,2] > 1,
           exp(-lambda_old[i] *
             ifelse(gp[i,2] == 3,
               d_phy[i,j],
               d_per_old[i,j])), 1)
    # Learning for Model 1
    v[i,j+1,1] <- ifelse(gp[i,1] != 1,
             ifelse(k[i,j] == 1,
               v[i,j,1] + alpha[i,1] *
                (r[i,j] - v[i,j,1]),
```

```
          v[i,j,1]), 0)
# Learning for Model 2
v[i,j+1,2] <- ifelse(gp[i,2] != 1,
        ifelse(k[i,j] == 1,
          v[i,j,2] + alpha[i,2] *
          (r[i,j] - v[i,j,2]),
          v[i,j,2]), 0)
}
# Initialize learned values
v[i,1,1] <- 0
v[i,1,2] <- 0
# Separate group assignments for each model
gp[i,1] ~ dcat(pi[,1])
gp[i,2] ~ dcat(pi[,2])
# Model 1 parameters (distribution-based perceptual)
lambda_1[i] ~ dnorm(mu_lambda[choice[i]],
      1/sigma_lambda[choice[i]]^2)T(overgen_limit,)
lambda_2[i] ~ dnorm(mu_lambda[2],
      1/sigma_lambda[2]^2)T(1e-9, overgen_limit)
lambda[i] <- ifelse(gp[i,1] == 1, 0,
      ifelse(gp[i,1] == 2, lambda_2[i], lambda_1[i]))
choice[i] <- ifelse(gp[i,1] == 3, 1, 2)
w0[i,1] ~ dnorm(w0_mu[1], 1/w0_sigma[1]^2)
w1_1[i,1] ~ dgamma(w1_a[1], w1_b[1])T(1,)
w1[i,1] <- ifelse(gp[i,1] == 1, 0, w1_1[i,1])
alpha_1[i,1] ~ dbeta(alpha_a[1], alpha_b[1])T(1e-9, 1-1e-9)
alpha[i,1] <- ifelse(gp[i,1] == 1, 0, alpha_1[i,1])
zn[i,1] <- ifelse(gp[i,1] == 1, 2, 1)
# Model 2 parameters (point-based perceptual)
lambda_old1[i] ~ dnorm(mu_lambda_old,
          1/sigma_lambda_old^2)T(overgen_limit,)
lambda_old2[i] ~ dnorm(mu_lambda_old,
          1/sigma_lambda_old^2)T(1e-9, overgen_limit)
lambda_old[i] <- ifelse(gp[i,2] == 1, 0,
          ifelse(gp[i,2] == 2, lambda_old2[i],
            lambda_old1[i]))
w0[i,2] ~ dnorm(w0_mu[2], 1/w0_sigma[2]^2)
w1_1[i,2] ~ dgamma(w1_a[2], w1_b[2])T(1,)
w1[i,2] <- ifelse(gp[i,2] == 1, 0, w1_1[i,2])
```

```
    alpha_1[i,2] ~ dbeta(alpha_a[2], alpha_b[2])T(1e-9, 1-1e-9)
    alpha[i,2] <- ifelse(gp[i,2] == 1, 0, alpha_1[i,2])
    zn[i,2] <- ifelse(gp[i,2] == 1, 2, 1)
  }
    # Model selection parameter
    pQ ~ dbeta(1,1)T(1e-9, 1-1e-9)
    # Hyperpriors for Model 1 (distribution-based perceptual)
    mu_lambda[1] ~ dlnorm(-2, pow(.5,-2))
    mu_lambda[2] ~ dlnorm(1, pow(.5,-2))
    sigma_lambda[1] ~ dunif(1e-9, .5)
    sigma_lambda[2] ~ dunif(1e-9, 5)
    alpha_a[1] <- alpha_mu[1] * alpha_kappa[1]
    alpha_b[1] <- (1 - alpha_mu[1]) * alpha_kappa[1]
    alpha_mu[1] ~ dbeta(1,1)T(1e-9, 1-1e-9)
    alpha_kappa[1] ~ dunif(1, 10)
    pi[1:4,1] ~ ddirch(c(1,1,1,1))
    w1_a[1] ~ dscaled.gamma(2,1)T(1e-9,)
    w1_b[1] ~ dscaled.gamma(2,1)T(1e-9,)
    w0_mu[1] ~ dnorm(0, 1/10^2)
    w0_sigma[1] ~ dscaled.gamma(2,1)
    sigma[1,1] ~ dunif(1e-9, 1.5)
    sigma[2,1] ~ dunif(1.5, 3)
    # Hyperpriors for Model 2 (point-based perceptual)
    mu_lambda_old ~ dlnorm(0, pow(.5,-2))
    sigma_lambda_old ~ dunif(1e-9, 1)
    alpha_a[2] <- alpha_mu[2] * alpha_kappa[2]
    alpha_b[2] <- (1 - alpha_mu[2]) * alpha_kappa[2]
    alpha_mu[2] ~ dbeta(1,1)T(1e-9, 1-1e-9)
    alpha_kappa[2] ~ dunif(1, 10)
    pi[1:4,2] ~ ddirch(c(1,1,1,1))
    w1_a[2] ~ dscaled.gamma(2,1)T(1e-9,)
    w1_b[2] ~ dscaled.gamma(2,1)T(1e-9,)
    w0_mu[2] ~ dnorm(0, 1/10^2)
    w0_sigma[2] ~ dscaled.gamma(2,1)
    sigma[1,2] ~ dunif(1e-9, 1.5)
    sigma[2,2] ~ dunif(1.5, 3)
}', file = "models/jags_models/super_model.txt")
```

30. Execute super-model MCMC sampling.
    a. Configure MCMC settings for stable convergence:

```
# Parameters to monitor
super_params <- c("pQ", "gp", "pi", "alpha_mu", "mu_lambda",
         "sigma_lambda", "w0_mu", "w0_sigma")
# Extended MCMC settings for stable Bayes factor estimation
super_mcmc <- list(
  n.chains = 4,
  n.iter = 250000, # Extended iterations for stable pQ estimation
  n.burnin = 100000,
  n.thin = 5, # Efficient storage
  n.cores = 4,
  parallel = TRUE
)
```

    b. Fit the super-model:

```
# Set reproducible seed
set.seed(2025)
# Fit super-model (this will take several hours)
super_model <- do.call(jags, c(
  list(data = super_data,
    model.file = "models/jags_models/super_model.txt",
    parameters.to.save = super_params),
  super_mcmc
))
# Check convergence
if(super_model$Rhat$pQ > 1.1) {
  warning("Poor convergence for pQ. Consider extending iterations.")
} else {
  cat("Convergence achieved for model selection parameter.\n")
}
```

31. Calculate Bayes factor using Savage-Dickey density ratio:

```
# Extract pQ samples (model selection parameter)
pQ_samples <- super_model$sims.list$pQ
# Fit posterior density using logspline
posterior_fit <- logspline(pQ_samples, lbound = 0, ubound = 1)
```

```
# Calculate Bayes factor using Savage-Dickey ratio
# H0: pQ = 0.5 (no difference between models)
# H1: pQ != 0.5 (models differ)
posterior_density_at_half <- dlogspline(0.5, posterior_fit)
prior_density_at_half <- 1 # Beta(1,1) density at 0.5
# Compute Bayes factor (H0/H1)
bayes_factor <- 1 / posterior_density_at_half
# Calculate posterior summaries
posterior_mean <- mean(pQ_samples)
posterior_median <- median(pQ_samples)
credible_interval <- quantile(pQ_samples, c(0.025, 0.975))
cat("\n=== MODEL COMPARISON RESULTS ===\n")
cat("Bayes Factor (H0/H1):", round(bayes_factor, 3), "\n")
# Interpret results
if (bayes_factor > 1) {
  if (posterior_mean > 0.5) {
    evidence_interpretation <- paste0(
      "Evidence FOR distribution-based perceptual model (BF = ",
      round(bayes_factor, 2),
      ") - larger BF = stronger evidence for model difference"
    )
  } else {
    evidence_interpretation <- paste0(
      "Evidence FOR point-based perceptual model (BF = ",
      round(bayes_factor, 2),
      ") - larger BF = stronger evidence for model difference"
    )
  }
} else {
  evidence_interpretation <- paste0(
    "Weak evidence for model difference (BF = ",
    round(bayes_factor, 2),
    " < 1) - models may be equivalent"
  )
}
```

32. Create visualizations and save results.
    a. Generate model comparison visualization:

```
# Create posterior distribution plot
pQ_df <- data.frame(pQ = pQ_samples)
model_comparison_plot <- ggplot(pQ_df, aes(x = pQ)) +
  geom_density(fill = "lightblue", alpha = 0.7, color = "black") +
  geom_vline(xintercept = 0.5, linetype = "dashed",
       color = "red", size = 1) +
  geom_vline(xintercept = posterior_mean, color = "blue", size = 1) +
annotate("text", x = 0.25, y = max(density(pQ_samples)$y) * 0.8,
    label = "Point-based\n(traditional)",
    hjust = 0.5, size = 3.5) +
  annotate("text", x = 0.75, y = max(density(pQ_samples)$y) * 0.8,
    label = "Distribution-based\n(probabilistic)",
    hjust = 0.5, size = 3.5) +
  annotate("text", x = 0.5, y = max(density(pQ_samples)$y) * 0.5,
    label = "H0: pQ = 0.5", hjust = 0.5, size = 3, color = "red") +
  labs(title = "Model Selection Parameter Posterior Distribution",
    subtitle = paste("BF =", round(bayes_factor, 2),
           "|", evidence_interpretation),
    x = "Model Selection Parameter (pQ)",
    y = "Posterior Density") +
  xlim(0, 1) + theme_minimal() +
  theme(plot.title = element_text(hjust = 0.5, face = "bold"),
    plot.subtitle = element_text(hjust = 0.5))
print(model_comparison_plot)
ggsave("outputs/figures/model_comparison_results.pdf",
    width = 10, height = 6, dpi = 300)
```

    b. Save results:

```
# Create results summary
model_comparison_summary <- list(
  bayes_factor = bayes_factor,
  evidence_interpretation = evidence_interpretation,
  pQ_posterior_mean = posterior_mean,
  pQ_posterior_median = posterior_median,
  pQ_credible_interval = credible_interval,
```

```
  analysis_date = Sys.Data(),
  convergence_rhat = super_model$Rhat$pQ
)
# Save all model comparison results
save(super_model, model_comparison_summary,
   bayes_factor, pQ_samples,
   file = "models/fitted_models/model_comparison_results.RData")
```

Note: This completes the computational workflow outlined in Figure 1, providing quantitative evidence for whether distribution-based or point-based perceptual assumptions better account for fear generalization behavior.

EXPECTED OUTCOMES

Upon successful completion of this protocol, researchers will obtain comprehensive insights into the computational mechanisms underlying fear generalization behavior, with quantitative estimates of both perceptual processing and generalization dynamics at individual and group levels.

The Bayesian perceptual model component yields trial-by-trial estimates of how participants transform physical stimuli into subjective perceptual representations. Individual participants will show distinct perceptual mapping functions characterized by intercept (β_0) and slope (β_1) parameters that quantify baseline perceptual magnitude and sensitivity to stimulus differences.

The forgetting rate parameter (ω) describes how quickly perceptual uncertainty decays over trials, revealing individual differences in Bayesian-like perceptual processes. Dynamic perceptual estimates reveal evolving uncertainty patterns through Kalman gain trajectories that typically start high during early trials, then gradually decrease and stabilize as participants develop confident perceptual representations.

The resulting distance matrices from stimulus distance computation capture stimulus similarity relationships, with distribution-based perceptual distances reflecting distributional overlap between uncertain perceptual representations rather than simple point differences.

The hierarchical mixture generalization model identifies four distinct pathways where learning rate (α) and generalization rate (λ) parameters reveal individual differences in fear acquisition and transfer: Non-Learners ($\alpha \approx 0$), Overgeneralizers (normal α but minimal λ causing broad fear responses), Physical Generalizers (exponential decay with λ governing physical distance-dependent generalization), and Perceptual Generalizers (exponential decay with λ following individual perceptual representations). These parameters quantify how effectively individuals acquire fear associations and how selectively they generalize across stimuli.

The model comparison framework produces quantitative evidence regarding which perceptual assumptions better account for generalization behavior through Bayes factor computation. The model selection parameter (β_M) exhibits posterior distributions shifted away from the neutral point of 0.5, with the degree and direction of shift reflecting evidence strength for either distribution-based or point-based perceptual assumptions. Changes in group allocation patterns between models reveal the extent to which accounting for perceptual uncertainty affects pathway classification, often showing shifts toward Perceptual Generalizer allocation when distribution-based perceptual distances better capture individual similarity computations.

The resulting parameter estimates provide actionable insights for both theoretical development and clinical applications, quantifying how perceptual processing differences contribute to individual variation in fear generalization and offering a principled framework for understanding when and why some individuals show more or less adaptive generalization patterns in fear-related contexts.

LIMITATIONS

This protocol has some limitations that researchers should carefully consider when applying these methods to their own research contexts.

The modeling framework makes several strong theoretical assumptions that may not hold across all experimental contexts or populations. The four-group mixture model assumes that all meaningful individual differences in generalization can be captured by these specific latent classes, potentially overlooking other important sources of heterogeneity or intermediate patterns.

Importantly, the thresholds used to define Non-Learners (learning rate ≈ 0) and Overgeneralizers (generalization rate producing >70% similarity at maximum stimulus distance) lack clinical validation. These computational boundaries, while theoretically motivated, have not been systematically evaluated against clinical measures. Future studies should investigate how these model-based classifications correspond to clinically meaningful distinctions.

The approach requires carefully controlled experimental designs with sufficient numbers of conditioning and generalization trials to provide adequate data for parameter estimation. The protocol may not be suitable for studies employing binary choice measures, or behavioral paradigms that do not involve explicit stimulus similarity judgments.

The requirement for trial-by-trial perceptual ratings and expectancy measures may also limit applicability to certain populations or experimental contexts where such detailed responses cannot be reliably obtained.

The super-model approach provides a theoretically principled method for Bayes factor computation. However, this approach requires extended MCMC chains to achieve stable estimation of the model selection parameter. Researchers must invest considerable time in convergence diagnostics and may need to re-run analyses with extended sampling if initial fits are inadequate.

TROUBLESHOOTING

This section provides practical solutions for common issues encountered during protocol implementation. Problems are linked to specific protocol steps where they are most likely to occur.

Problem 1

JAGS installation fails or jagsUI cannot connect (related to Software environment and technical setup, step 4).

While JAGS installation is typically straightforward, occasional issues may arise with system PATH configuration or R-JAGS interface setup. Symptoms include error messages like "Failed to locate any version of JAGS" or compilation failures when testing the verification model. Users may also encounter connection errors when R attempts to interface with JAGS through the jagsUI package, particularly due to version incompatibility or incorrect path configuration.

Potential solution

- Verify version compatibility: Recent versions of R (4.2.0 or later) require JAGS 4.3.1 or later. Check your versions using `R.version.string` and test JAGS installation with `Sys.which("jags")` in R, which should return the path to the JAGS executable.
- For Windows users experiencing "Failed to locate any version of JAGS" errors, set the `JAGS_HOME` environment variable:

```
Sys.setenv(JAGS_HOME="C:/Program Files/JAGS/JAGS-4.3.0")
```

The exact path should match your actual JAGS installation directory.

- After setting the environment variable, restart R and test the connection.
- For macOS users, ensure JAGS is installed in the standard location (/usr/local) and verify compatibility with your macOS version.
- If jagsUI continues to have connection issues, try the `r2jags` package as an alternative interface.

Problem 2

NaN or infinite values in distance computation (related to Stimulus distance computation, step 15).

Distance computation produces invalid values (NaN, Inf) due to extremely small standard deviations in the perceptual distributions. When standard deviations approach zero, the `dnorm()` function returns extremely large density values that cause numerical overflow when multiplied by the integration range width in the Monte Carlo calculation.

Potential solution

- Prevent this issue at the source by ensuring proper truncation bounds in the Bayesian perceptual model specification. In the JAGS model file, ensure that all variance parameters have appropriate lower bounds (e.g., `T(1e-3,)`) to prevent extremely small standard deviations from being sampled.
- Alternatively, if working with pre-fitted model results, implement protective lower bounds for all standard deviation parameters before distance calculation:

```
sigma_psi <- pmax(sigma_psi, 1e-3)

cs_mem_sd <- pmax(cs_mem_sd, 1e-3)
```

- Apply this constraint immediately after extracting the Bayesian perceptual model results and before calling the distance computation function. The model-level truncation approach is recommended as it addresses the root cause.

Problem 3

Label switching in mixture generalization model (related to Hierarchical mixture generalization model, step 22).

Mixture models can suffer from label switching where MCMC chains arbitrarily reassign participants between latent groups across iterations, creating erratic trace plots for group probability parameters (π). Rather than forcing artificial solutions, this often indicates genuine uncertainty about group membership for some participants.

Potential solution

- Implement a certainty-based group assignment approach that acknowledges individual differences in learning and generalization patterns. After model fitting, calculate the proportion of posterior samples assigning each participant to each group:

```
group_certainty <- apply(generalization_model$sims.list$gp, 2,
          function(x) { max(table(x)) / length(x) })
```

- Use an 80% certainty threshold as the recommended starting point:

```
certainty_threshold <- 0.80
final_groups <- ifelse(group_certainty > certainty_threshold,
  apply(generalization_model$mean$gp, 1, which.max),
  5) # Group 5 = Unknown
```

- Participants classified as "Unknown" represent genuine heterogeneity requiring further investigation rather than model failure.
- If more than 20% are "Unknown," consider lowering the threshold to 70%. If fewer than 10% are "Unknown," consider raising it to 90% for more stringent classification.

Problem 4

Memory limitations during MCMC fitting (related to Bayesian perceptual model, Hierarchical mixture generalization model, and Model comparison framework).

Large datasets or extended MCMC chains may exceed available memory, particularly during model comparison framework fitting with multiple likelihood calculations. Memory issues typically manifest as R crashes, "cannot allocate vector" errors, or system slowdowns when available RAM is exhausted.

Potential solution

- Monitor memory usage throughout analysis:

```
library(pryr)
cat("Current R memory:", round(pryr::mem_used()/1024^3, 2), "GB\n")
```

- Reduce monitored parameters during fitting:

```
# Essential parameters only (exclude y_pred initially)
params_essential <- c("gp", "pi", "alpha", "lambda", "w0", "w1", "sigma")
```

- Clear memory between major analysis steps:

```
rm(large_data_object, intermediate_results)
gc()
```

- Consider session separation for memory-intensive analyses—fit models in separate R sessions and save results between steps:

```
# Session 1: Perceptual model fitting

source("01_perceptual_model.R")

save(perceptual_results, file = "perceptual_fitted.RData")

# Session 2: Generalization model fitting

load("perceptual_fitted.RData")

source("02_generalization_model.R")
```

- Use conservative parallel processing setup: `safe_cores <- min(2, detectCores() - 1)`

Problem 5

Non-convergence of MCMC chains (related to Bayesian perceptual model, step 11, and Hierarchical mixture generalization model, step 22).

Non-convergence occurs when MCMC chains fail to reach the stationary distribution, indicated by $\widehat{R} > 1.1$, divergent trace plots, or chains that remain stuck in different regions of parameter space. Common causes include insufficient iterations, poor initial values, weakly identified parameters, or problematic posterior geometry (e.g., "funnel" shapes in hierarchical models).

Potential solution

- Diagnose the source of non-convergence before attempting fixes. Identify which parameters are problematic:

```
rhat_values <- model$Rhat

problematic <- names(rhat_values)[rhat_values > 1.1]

library(bayesplot)

mcmc_trace(model$samples, pars = problematic[1:min(4, length(problematic))])
```

- Increase iterations and burn-in if trace plots show chains that are mixing but have not yet stabilized:

```
mcmc_settings <- list(n.chains = 4, n.iter = 200000,

          n.burnin = 150000, n.thin = 25)
```

- Improve initial values by providing dispersed but reasonable starting points within the prior support:

```
generate_inits <- function() {

  list(alpha_mu = runif(1, 0.3, 0.7), alpha_kappa = runif(1, 2, 8),

    lambda_mu = rlnorm(1, 0, 0.3), w0_mu = rnorm(1, 0, 2))

}

inits_list <- lapply(1:4, function(i) generate_inits())
```

- Apply non-centered parameterization (Matt trick) when population and individual parameters show strong posterior correlations ("funnel" shapes). For a tutorial, see Baribault and Collins (2025)[23]:

```
# Non-centered (improved mixing):

# w0_raw[i] ~ dnorm(0, 1)

# w0[i] <- w0_mu + w0_sigma * w0_raw[i]
```

- Revise priors or simplify the model if convergence issues persist. Note that changing priors or model structure constitutes a change to the model itself.[24] Such revisions should be theoretically justified, clearly documented, accompanied by sensitivity analysis, and acknowledged when interpreting results.

RESOURCE AVAILABILITY

Lead contact

Further information and requests for resources should be directed to and will be fulfilled by the lead contact, Kenny Yu (kenny.yu@kuleuven.be).

Technical contact

Technical questions on executing this protocol should be directed to and will be answered by the technical contact, Kenny Yu (kenny.yu@kuleuven.be).

Materials availability

All computational materials associated with this protocol are openly available through the Open Science Framework repository at https://osf.io/a5xyg/, where materials from Yu et al.[2] are deposited. This includes JAGS model files, R analysis scripts, and datasets that demonstrate the complete computational pipeline. No restrictions apply to the use of these computational resources for academic research purposes. The protocol utilizes open-source software (R, JAGS) and publicly available R packages, ensuring full reproducibility without proprietary software requirements.

Data and code availability

Complete computational code and example datasets necessary to reproduce the analyses described in this protocol are available at https://osf.io/a5xyg/. This repository contains comprehensive JAGS model specifications for the perceptual model, generalization model, and super-model comparison framework. R scripts for data preparation, model fitting, and result visualization are included with detailed documentation.

ACKNOWLEDGMENTS

K.Y. is supported by a Research Foundation Flanders (FWO) research project (G079520N). J.Z. is a Postdoctoral Research Fellow of FWO (12P8623N) and received funding from the Alexander von Humboldt Stiftung. K.Y., W.V., and F.T. are also supported in part by the Research Fund of KU Leuven (C14/23/062). The resources and services used in this work were also provided by the VSC (Flemish Supercomputer Center), funded by FWO and the Flemish Government. The funders had no role in study design, data collection and analysis, decision to publish, or preparation of the manuscript.

AUTHOR CONTRIBUTIONS

Conceptualization, K.Y., W.V., F.T., and J.Z.; methodology, K.Y., W.V., F.T., and J.Z.; software, K.Y.; investigation, K.Y.; resources, K.Y.; visualization, K.Y.; writing – original draft, K.Y.; writing – review and editing, K.Y., W.V., F.T., and J.Z.; funding acquisition, J.Z. (G079520N) and F.T. (C14/23/062).

DECLARATION OF INTERESTS

The authors declare no competing interests.

REFERENCES

1. Yu, K., Tuerlinckx, F., Vanpaemel, W., and Zaman, J. (2023). Humans display interindividual differences in the latent mechanisms underlying fear generalization behaviour. Commun. Psychol. *1*, 5. https://doi.org/10.1038/s44271-023-00005-0. https://www.nature.com/articles/s44271-023-00005-0.

2. Yu, K., Vanpaemel, W., Tuerlinckx, F., and Zaman, J. (2025). The probabilistic and dynamic nature of perception in human generalization behavior. iScience *28*, 112228. https://doi.org/10.1016/j.isci.2025.112228. https://linkinghub.elsevier.com/retrieve/pii/S2589004225004894.

3. Zaman, J., Chalkia, A., Zenses, A.-K., Bilgin, A.S., Beckers, T., Vervliet, B., and Boddez, Y. (2021). Perceptual variability: Implications for learning and generalization. Psychon. Bull. Rev. *28*, 1–19. https://doi.org/10.3758/s13423-020-01780-1.

4. Yu, K., Verheyen, S., and Zaman, J. (2025). Beyond dichotomies in generalization

research: A reply to Lee and Schlegelmilch (2025). J. Exp. Psychol. Gen. *154*, 1176–1181. https://doi.org/10.1037/xge0001732.

5. Gelman, A., and Rubin, D.B. (1992). Inference from Iterative Simulation Using Multiple Sequences. Stat. Sci. *7*. https://doi.org/10.1214/ss/1177011136. https://projecteuclid.org/journals/statistical-science/volume-7/issue-4/Inference-from-Iterative-Simulation-Using-Multiple-Sequences/10.1214/ss/1177011136.full.

6. Wagenmakers, E.-J., Marsman, M., Jamil, T., Ly, A., Verhagen, J., Love, J., Selker, R., Gronau, Q.F., Šmíra, M., Epskamp, S., et al. (2018). Bayesian inference for psychology. Part I: Theoretical advantages and practical ramifications. Psychon. Bull. Rev. *25*, 35–57. https://doi.org/10.3758/s13423-017-1343-3.

7. Kruschke, J.K., and Liddell, T.M. (2018). The Bayesian New Statistics: Hypothesis testing, estimation, meta-analysis, and power analysis from a Bayesian perspective. Psychon. Bull. Rev. *25*, 178–206. https://doi.org/10.3758/s13423-016-1221-4.

8. Wagenmakers, E.-J., Lodewyckx, T., Kuriyal, H., and Grasman, R. (2010). Bayesian hypothesis testing for psychologists: A tutorial on the Savage–Dickey method. Cogn. Psychol. *60*, 158–189. https://doi.org/10.1016/j.cogpsych.2009.12.001. https://linkinghub.elsevier.com/retrieve/pii/S0010028509000826.

9. Kruschke, J.K. (2021). Bayesian Analysis Reporting Guidelines. Nat. Hum. Behav. *5*, 1282–1291. https://doi.org/10.1038/s41562-021-01177-7. https://www.nature.com/articles/s41562-021-01177-7.

10. Lee, M.D. (2011). How cognitive modeling can benefit from hierarchical Bayesian models. J. Math. Psychol. *55*, 1–7. https://doi.org/10.1016/j.jmp.2010.08.013. https://linkinghub.elsevier.com/retrieve/pii/S0022249610001148.

11. Pavlov, I.P. (1927). Conditioned reflexes: an investigation of the physiological activity of the cerebral cortex. Conditioned Reflexes: An Investigation of the Physiological Activity of the Cerebral Cortex (Oxford, England: Oxford Univ. Press), p. 430.

12. Brooks, S.P., and Gelman, A. (1998). General Methods for Monitoring Convergence of Iterative Simulations. J. Comput. Graph Stat. *7*, 434–455. https://doi.org/10.1080/10618600.1998.10474787. http://www.tandfonline.com/doi/abs/10.1080/10618600.1998.10474787.

13. Vehtari, A., Gelman, A., Simpson, D., Carpenter, B., and Bürkner, P.-C. (2021). Rank-Normalization, Folding, and Localization: An Improved R ˆ for Assessing Convergence of MCMC (with Discussion). Bayesian Anal. *16*. https://doi.org/10.1214/20-BA1221. https://projecteuclid.org/journals/bayesian-analysis/volume-16/issue-2/Rank-Normalization-Folding-and-Localization–An-Improved-R%cb%86-for/10.1214/20-BA1221.full.

14. Gelman, A., Vehtari, A., Simpson, D., Margossian, C.C., Carpenter, B., Yao, Y., Kennedy, L., Gabry, J., Bürkner, P.-C., and Modrák, M. (2020). Bayesian Workflow. arXiv. https://doi.org/10.48550/arXiv.2011.01808. http://arxiv.org/abs/2011.01808.ArXiv:2011.01808.

15. Petzschner, F.H., Glasauer, S., and Stephan, K.E. (2015). A Bayesian perspective on magnitude estimation. Trends Cogn. Sci. *19*, 285–293. https://doi.org/10.1016/j.tics.2015.03.002. https://linkinghub.elsevier.com/retrieve/pii/S1364661315000509.

16. Ma, W.J., Beck, J.M., Latham, P.E., and Pouget, A. (2006). Bayesian inference with probabilistic population codes. Nat. Neurosci. *9*, 1432–1438. https://doi.org/10.1038/nn1790.

17. Ma, W.J. (2019). Bayesian Decision Models: A Primer. Neuron *104*, 164–175. https://doi.org/10.1016/j.neuron.2019.09.037.

18. Yu, K., Lin, T.-Y., Zaman, J., Tuerlinckx, F., and Vanhasbroeck, N. (2025). Consistency of perceptual response variability in size estimation and reproduction tasks. Behav. Res. Methods *57*, 127. https://doi.org/10.3758/s13428-025-02650-1.

19. Yu, K., Vanpaemel, W., Tuerlinckx, F., and Zaman, J. (2024). The representational instability in the generalization of fear learning. NPJ Sci. Learn. *9*, 78. https://doi.org/10.1038/s41539-024-00287-x. https://www.nature.com/articles/s41539-024-00287-x.

20. Gelman, A. (2006). Prior distributions for variance parameters in hierarchical models (comment on article by Browne and Draper). Bayesian Anal. *1*. https://doi.org/10.1214/06-BA117A. https://projecteuclid.org/journals/bayesian-analysis/volume-1/issue-3/Prior-distributions-for-variance-parameters-in-hierarchical-models-comment-on/10.1214/06-BA117A.full.

21. Rescorla, R.A. (1976). Stimulus generalization: Some predictions from a model of Pavlovian conditioning. J. Exp. Psychol. Anim. Behav. Process. *2*, 88–96. https://doi.org/10.1037/0097-7403.2.1.88.

22. Shepard, R.N. (1987). Toward a Universal Law of Generalization for Psychological Science. Science *237*, 1317–1323. https://doi.org/10.1126/science.3629243. https://www.science.org/doi/10.1126/science.3629243.

23. Baribault, B., and Collins, A.G.E. (2025). Troubleshooting Bayesian cognitive models. Psychol. Methods *30*, 128–154. https://doi.org/10.1037/met0000554.

24. Gelman, A. (2004). Parameterization and Bayesian Modeling. J. Am. Stat. Assoc. *99*, 537–545. https://doi.org/10.1198/016214504000000458.