Image Inference (Experiment 3)

Preprocessing

(The "WorkerId" column within the MTurk results file has been modified to maintain worker privacy.)

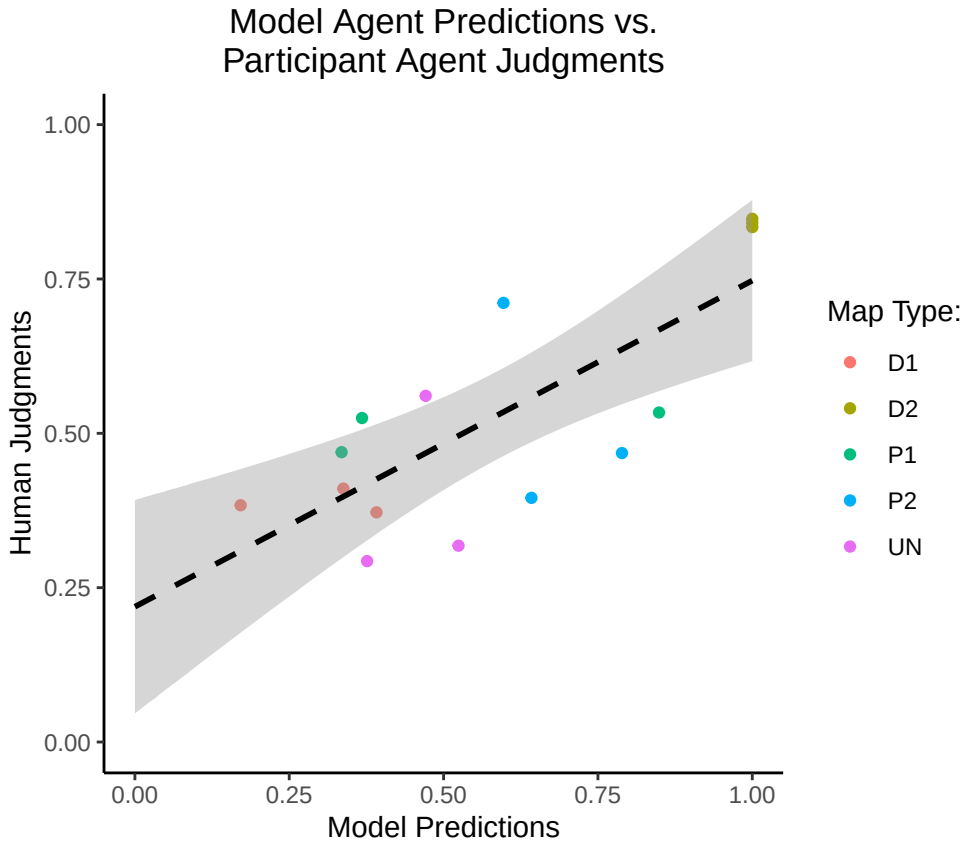
```
# Read in the participant data (after manually removing errors).
data_0 = read_csv(file.path(human_path, "raw_data.csv"), quote="~")

# Read in the MTurk results file.
mturk_results = read_csv(file.path(human_path, "mturk_results.csv"),
                         col_names=TRUE) %>%
  mutate(Answer.surveycode=substr(Answer.surveycode, 3,
                                  length(Answer.surveycode))) %>%
  filter(AssignmentStatus=="Approved")

# Stop if there are duplicate database entries based on the MTurk results file.
duplicates = setdiff(data_0$unique_id, mturk_results$Answer.surveycode)
if (length(duplicates) != 0) { stop("There are duplicate entries.") }

# Convert the JSON string into JSON.
data_1 = lapply(data_0$results, fromJSON)

# Extract the trial information for each participant and stack them.
age = c()
data_3 = tibble()
for (p in 1:length(data_1)) {
  # Trim the map and add the participant ID back in.
  data_2 = data_1[p][[1]]$trials %>%
    as.data.frame() %>%
    mutate(map=gsub(".png", "", map), unique_id=data_0$unique_id[p],
           wrong_attempts=data_1[p][[1]]$catch_trials$wrong_attempts)

  # Extract and store this participant's age.
  age = c(age, as.integer(data_1[p][[1]]$subject_information$age))

  # Stack the trial information for the current participant.
  data_3 = rbind(data_3, data_2)
}

# Write the preprocessed data.
write_csv(data_3, file.path(human_path, "data.csv"))
```

Number of Agents Inference

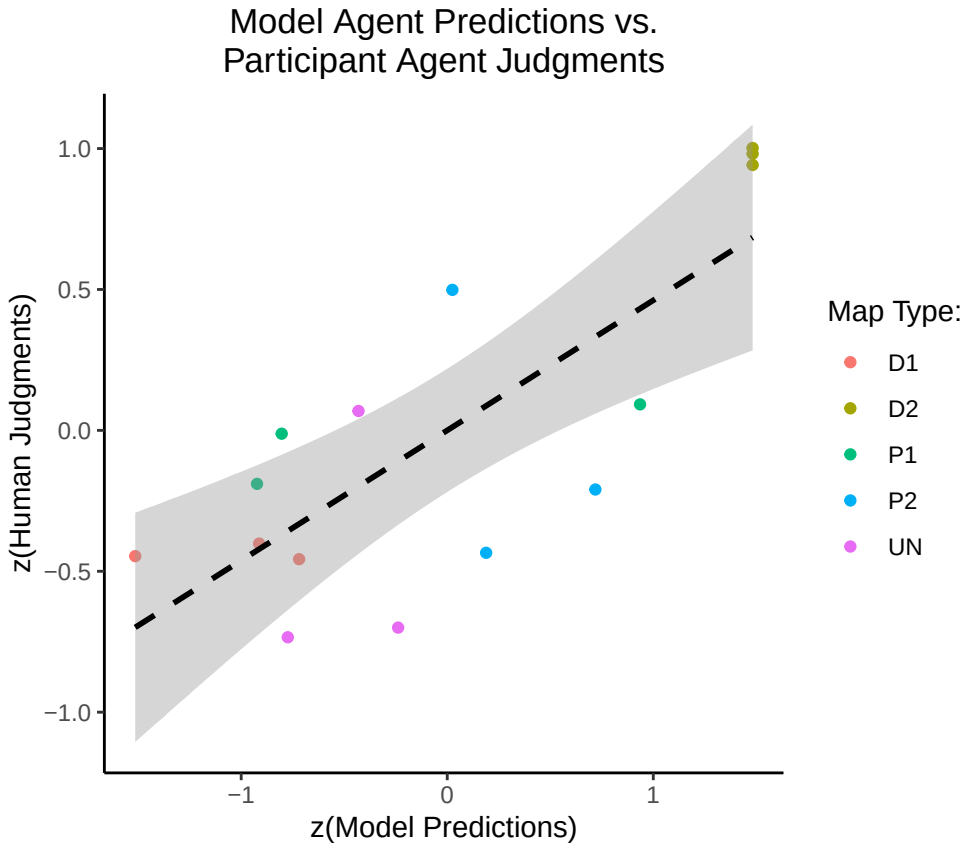
Here we generate scatter plots comparing participant judgments (N=40; M=37.62 years, SD=11.94 years) against our model predictions on how many agents were in each room given the observed physical evidence.

```
# Read in the preprocessed data.
data_3 = read_csv(file.path(human_path, "data.csv"))

# Compute the z-scored participant judgments.
data_4 = data_3 %>%
  group_by(unique_id) %>%
  mutate(z_num_agents=scale(num_agents)[,1]) %>%
  ungroup()

# Define the bootstrap function for the bootstrap statistic.
compute_mean = function(data, indices) {
  return(mean(data[indices]))
}

# Define the bootstrap function to simulate the data.
compute_bootstrap = function(data) {
  # Run the simulations.
  simulations = boot(data=data,
                     statistic=compute_mean,
                     R=10000)

  return(boot.ci(simulations, type="bca")$bca)
}

# Compute the bootstrapped 95% CIs.
set.seed(seed)
ci = data.frame()
for (m in unique(data_3$map)) {
  # Filter the current map.
  data_5 = data_4 %>%
    filter(map==m)

  # Compute the bootstrap for the dependent measure.
  num_agents_bootstrap = compute_bootstrap(data_5$num_agents)
  z_num_agents_bootstrap = compute_bootstrap(data_5$z_num_agents)

  # Store the bootstrapped 95% CIs.
  ci = rbind(ci, data.frame(map=m,
                            lower=num_agents_bootstrap[4],
                            upper=num_agents_bootstrap[5],
                            z_lower=z_num_agents_bootstrap[4],
                            z_upper=z_num_agents_bootstrap[5]))
}

# Read in the goal predictions.
model_0 = read_csv(file.path(model_path, "data.csv"))

# Perform some basic preprocessing and then z-score the model predictions.
model_1 = model_0 %>%
```

```
  select(-l_1, -l_2) %>%
  rename(model=p_2) %>%
  mutate(z_model=scale(model)[,1])

# Merge the non-standardized and standardized participant judgments with the
# bootstrapped 95% CIs and the model predictions.
data_6 = data_4 %>%
  rename(human=num_agents, z_human=z_num_agents) %>%
  group_by(map) %>%
  summarize(mean_human=mean(human), mean_z_human=mean(z_human)) %>%
  left_join(ci) %>%
  left_join(model_1)

# Generate a plot of the goal comparison using the non-standardized model
# predictions and mean participant judgments.
plot_0 = data_6 %>%
  ggplot(aes(x=model, y=mean_human, label=map)) +
  geom_point(aes(color=substr(map, 0, 2))) +
  geom_smooth(method="lm", se=TRUE, linetype="dashed", color="black",
              fullrange=TRUE) +
  ggtitle("Model Agent Predictions vs.\nParticipant Agent Judgments") +
  xlab("Model Predictions") +
  ylab("Human Judgments") +
  xlim(0.0, 1.0) +
  ylim(0.0, 1.0) +
  scale_color_discrete(name="Map Type:") +
  theme_classic() +
  theme(aspect.ratio=1.0,
        plot.title=element_text(hjust=0.5),
        legend.title=element_text(hjust=0.5))
plot_0
```

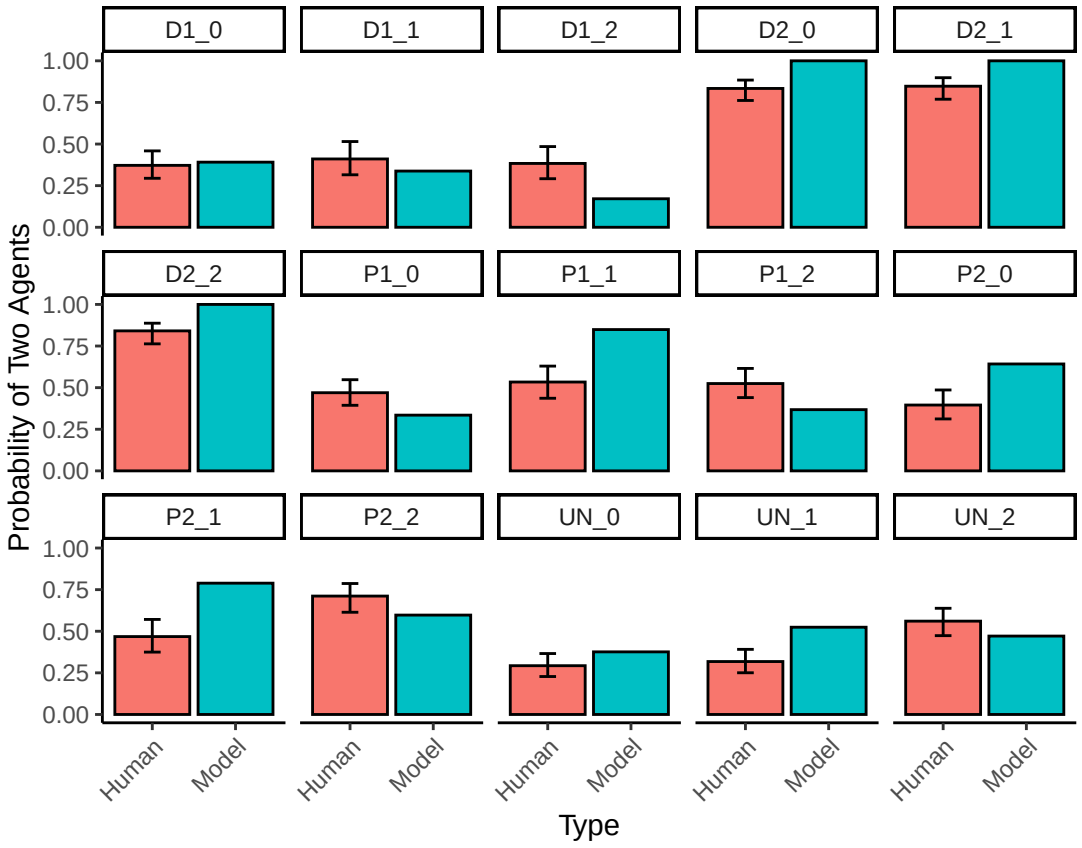



```
# Generate a plot of the goal comparison using the standardized model
# predictions and mean participant judgments.
plot_1 = data_6 %>%
  ggplot(aes(x=z_model, y=mean_z_human, label=map)) +
  geom_point(aes(color=substr(map, 0, 2))) +
  geom_smooth(method="lm", se=TRUE, linetype="dashed", color="black") +
  ggtitle("Model Agent Predictions vs.\nParticipant Agent Judgments") +
  xlab("z(Model Predictions)") +
  ylab("z(Human Judgments)") +
  scale_color_discrete(name="Map Type:") +
  theme_classic() +
  theme(aspect.ratio=1.0,
        plot.title=element_text(hjust=0.5),
        legend.title=element_text(hjust=0.5))
plot_1
```

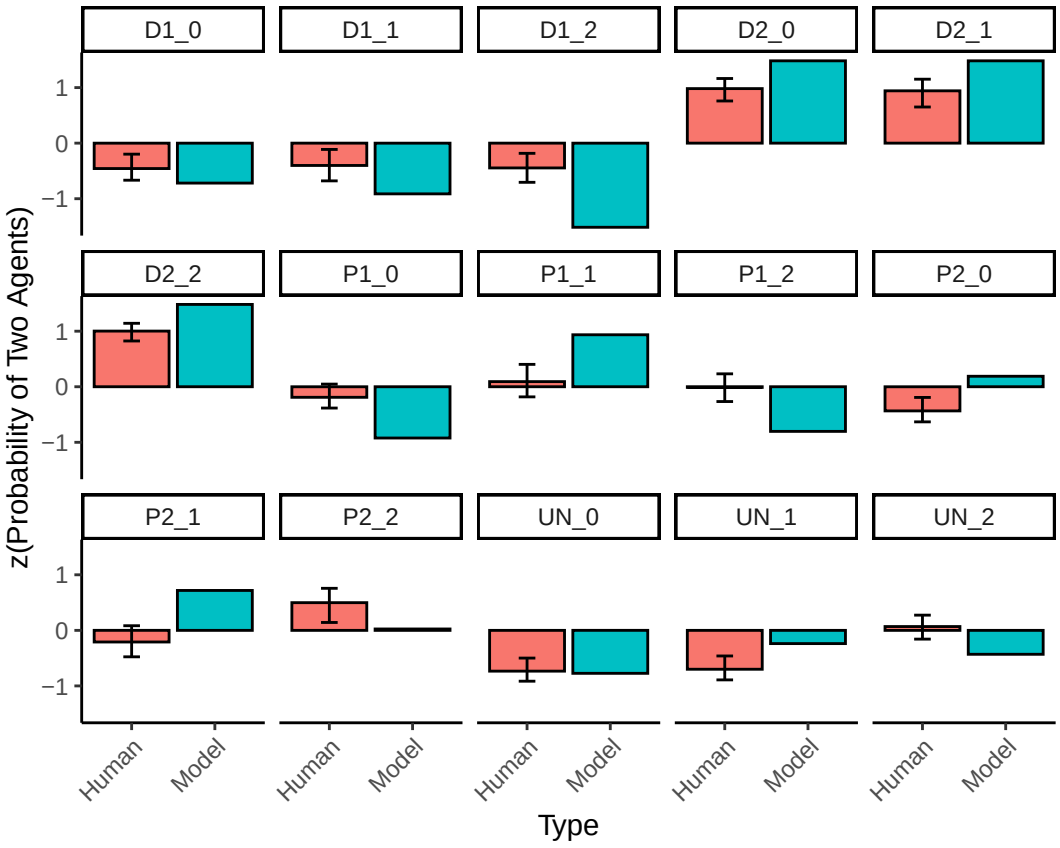


The Pearson correlation using the the non-standardized data is r=0.76 (95% CI: 0.43-0.91). The Pearson correlation using the standardized data is r=0.77 (95% CI: 0.45-0.91). Next, we generate comparisons for each trial for both groups of data.

```
# Generate a plot of agent inferences by trial using the non-standardized
# model predictions and mean participant judgments.
plot_2 = data_6 %>%
  gather(type, value, mean_human, model) %>%
  mutate(lower=ifelse(type=="model", NA, lower),
         upper=ifelse(type=="model", NA, upper)) %>%
  ggplot(aes(x=type, y=value)) +
  geom_bar(aes(fill=type), color="black", stat="identity") +
  geom_errorbar(aes(ymin=lower, ymax=upper), width=0.2) +
  facet_wrap(~map, nrow=3) +
  scale_x_discrete(name="Type",
                   limits=c("mean_human", "model"),
                   labels=c("Human", "Model")) +
  ylab("Probability of Two Agents") +
  theme_classic() +
  theme(aspect.ratio=1.0,
        legend.position="none",
        axis.text.x=element_text(hjust=1.0, angle=45))
plot_2
```




```
# Generate a plot of agent inferences by trial using the standardized
# model predictions and mean participant judgments.
plot_3 = data_6 %>%
  gather(type, value, mean_z_human, z_model) %>%
  mutate(z_lower=ifelse(type=="z_model", NA, z_lower),
         z_upper=ifelse(type=="z_model", NA, z_upper)) %>%
  ggplot(aes(x=type, y=value)) +
  geom_bar(aes(fill=type), color="black", stat="identity") +
  geom_errorbar(aes(ymin=z_lower, ymax=z_upper), width=0.2) +
  facet_wrap(~map, nrow=3) +
  scale_x_discrete(name="Type",
                   limits=c("mean_z_human", "z_model"),
                   labels=c("Human", "Model")) +
  ylab("z(Probability of Two Agents)") +
  theme_classic() +
  theme(aspect.ratio=1.0,
        legend.position="none",
        axis.text.x=element_text(hjust=1.0, angle=45))
plot_3
```



Supplementary analyses

Predicting goal inference as a function of goal and entrance distance

Now we test whether an alternative model that uses simple distance heuristics performs better than our model. This alternative model consists of a logistic regression that attempts to predict how many agents were in the room using (1) the distance between each pile of cookie crumbs and each goal, (2) the average distance between each pile of cookie crumbs and the entrances, (3) the number of entrances, and (4) all of their interactions.

```
# Read in the distances between the observation and each goal and entrance.
data_7 = read_csv("data/experiment_3/alternative/distances.csv")

# Stitch the predictors to the non-standardized and standardized mean
# participant judgments.
data_8 = data_6 %>%
  select(map, mean_human, mean_z_human) %>%
  left_join(data_7)
data_8 = data_6 %>%
  select(map, mean_human, mean_z_human) %>%
  left_join(data_7) %>%
  # mutate(mean_human=round(mean_human))
  mutate(mean_human=ifelse(mean_human>=0.5, 2, 1))
```

```
# Set up the alternative model.
x = model.matrix(mean_human~A_d1*B_d1*C_d1*A_d2*B_d2*C_d2*entrance_d1
                 *entrance_d2*num_entrances, data_8)[,-1]
y = data_8$mean_human

# Train the alternative model (using LASSO regularization to avoid overfitting)
# and generate predictions using the leave-one-out method.
set.seed(seed)
predictions = c()
for (testing_trial in 1:length(data_8$map)) {
  # Set up an array of random regularization parameters.
  lambdas = 10^seq(10, -2, length=100)

  # Partition the data according to the leave-one-out method.
  training_trials = setdiff(1:length(data_8$map), testing_trial)
  x_train = x[training_trials,]
  y_train = y[training_trials]
  x_test = x[testing_trial,]
  y_test = y[testing_trial]

  # Train the model and find the lambda parameter with the best fit.
  lasso_model = glmnet(x_train, y_train, family="binomial", alpha=1,
                       lambda=lambdas)
  cv_results = cv.glmnet(x_train, y_train, family="binomial", alpha=1)
  best_lambda = cv_results$lambda.min

  # Generate and store the prediction using the test data.
  predictions = c(predictions,
                  predict(lasso_model, s=best_lambda, newx=t(x_test),
                          family="binomial", type="response"))
}

# Store the predictions of the alternative model.
alternative = data.frame(
  map=data_8$map,
  predictions=predictions
)
```

```
# Stitch the alternative model predictions to the mean participant judgments and our model predictions.
data_9 = data_6 %>%
  select(map, mean_human, model) %>%
  left_join(alternative) %>%
  rename(alternative=predictions)

# Define the bootstrap function for the bootstrap statistic.
compute_cor = function(data, indices) {
  return(cor(data$alternative[indices], data$mean_human[indices],
             method="pearson"))
}

# Define the bootstrap function to simulate the data.
compute_bootstrap = function(data) {
  # Run the simulations.
```

```
  simulations = boot(data=data,
                     statistic=compute_cor,
                     R=10000)

  return(boot.ci(simulations, type="bca")$bca)
}

# Compute the bootstrapped 95% CI.
set.seed(seed)
cor_2 = cor(data_9$alternative, data_9$mean_human, method="pearson")
cor_2_bootstrap = compute_bootstrap(data_9)
cor_2_ci = data.frame(
  lower=cor_2_bootstrap[4],
  upper=cor_2_bootstrap[5]
)

# Define the bootstrap function for the bootstrap statistic.
compute_cor_diff = function(data, indices) {
  cor_alternative = cor(data$alternative[indices], data$mean_human[indices],
                        method="pearson")
  cor_model = cor(data$model[indices], data$mean_human[indices],
                  method="pearson")
  return(cor_model-cor_alternative)
}

# Define the bootstrap function to simulate the data.
compute_bootstrap = function(data) {
  # Run the simulations.
  simulations = boot(data=data,
                     statistic=compute_cor_diff,
                     R=10000)

  return(boot.ci(simulations, type="bca")$bca)
}

# Compute the correlation difference and the bootstrapped 95% CI.
set.seed(seed)
cor_3 = cor_0 - cor_2
cor_3_bootstrap = compute_bootstrap(data_9)
cor_3_ci = data.frame(
  lower=cor_3_bootstrap[4],
  upper=cor_3_bootstrap[5]
)
```

The Pearson correlation of the alternative model is r=0.19 (95% CI: -0.3-0.66). The correlation difference between the alternative model and our model is Δr=0.58 (95% CI: 1.17-0.12).

Analyzing model error as a function of trial type

Here we measure the model error in each of the three primary trial types: definitely (D) trials, probably (P) trials, and uncertain (U) trials.

```r
# Compute the model error on each trial, categorize the trials by trial type,
# then compute the mean squared error (MSE) as a function of trial type.
data_10 = data_6 %>%
  mutate(squared_model_error=(mean_human-model)**2,
         trial_type=substr(map, 1, 1)) %>%
  select(map, trial_type, squared_model_error) %>%
  group_by(trial_type) %>%
  summarize(mse=mean(squared_model_error))

# Plot the MSE.
data_10 %>%
  ggplot(aes(x=trial_type, y=mse, fill=trial_type)) +
  geom_bar(color="black", stat="identity") +
  xlab("Trial Type") +
  ylab("MSE") +
  scale_fill_discrete(name="Trial Type:") +
  theme_classic() +
  theme(aspect.ratio=1.0,
        legend.title=element_text(hjust=0.5))
```