Contents

Part I

Speech

1

Mice Can Learn Phonetic Categories

We perceive speech as a series of relatively invariant phonemes despite extreme variability in the acoustic signal. To be perceived as nearly-identical phonemes, speech sounds that vary continuously over a range of acoustic parameters must be perceptually discretized by the auditory system. Such many-to-one mappings of undifferentiated sensory information to a finite number of discrete categories are ubiquitous in perception. Although many mechanistic models of phonetic perception have been proposed, they remain largely unconstrained by neurobiological data. Current human neurophysiological methods lack the necessary spatiotemporal resolution to provide it: speech is too fast and the neural circuitry involved is too small. Here we demonstrate that mice are capable of learning generalizable phonetic categories, and can thus serve as a model for phonetic perception. Mice learned to discriminate consonants, and generalized consonant identity across novel vowel contexts and speakers, consistent with true category learning. A mouse model, given the powerful genetic and electrophysiological tools for probing neural circuits available for them, has the potential to powerfully augment our mechanistic understanding of phonetic perception.

1.1 Introduction

Lack of acoustic invariance in phonemes

We perceive speech as a series of relatively invariant phonemes despite extreme variability in the acoustic signal. This lack of order within phonemic categories remains one of the fundamental problems of speech perception [1]. Plosive stop consonants (such as /b/ or /g/) are the paradigmatic example of phonemes with near-categorical perception [2, 3, 4] without invariant acoustic structure [5, 6]. The problem is not just that phonemes are acoustically variable, but rather that there is a fundamental lack of invariance in the relation between phonemes and the acoustic signal [6]. Despite our inability to find a source of invariance in the speech signal, the auditory system learns some acoustic-perceptual mapping such that a plosive stop like /b/ is perceived as nearly identical across phonetic contexts. A key source of variability is coarticulation, which causes the sound of a spoken consonant to be strongly affected by neighboring segments, such as vowels. Coarticulation occurs during stop production because the articulators (such as the tongue or lips) have not completely left the positions from the preceding phoneme, and are already moving to anticipate the following phoneme [7, 8]. Along with many other sources of acoustic variation like speaker identity, sex, accent, or environmental noise; coarticulation guarantees that a given stop consonant does not have a uniquely invariant acoustic structure across phonetic contexts. In other words, there is no canonical acoustic /b/ [7, 2]. Phonetic perception therefore cannot be a simple, linear mapping of some continuous feature space to a discrete phoneme space. Instead it requires a mapping that flexibly uses evidence from multiple, imperfect cues depending on context [2, 9].

This invariant perception of phonemes, despite extreme variability in the physical speech signal, is referred to as the non-invariance problem [10].

Generality of phonetic perception

The lack of a simple mapping between acoustic attributes and phoneme identity has had a deep influence on phonetics, in part motivating the hypothesis that speech is mechanistically unique to humans [11], and the development of non-acoustic theories of speech perception (most notably motor theories [9, 7, 12]). However, it has been clear for more than 30 years that at least some auditory components of speech perception are not unique to humans, suggesting that human speech perception exploits evolutionarily-preserved functions of the auditory system [6, 13, 14, 15]. For example, nonhuman animals like quail [6, 16], chinchillas [17], rats [18], macaques [19], and songbirds [20] are capable of learning phonetic categories that share some perceptual qualities with humans [**?** 21]. This is consistent with the idea that categorizing phonemes is just one instance of a more general problem faced by all auditory systems, which typically extract useable information from complex acoustic environments by reducing them to a small number of 'auditory objects' (for review, see [22]).

Neurolinguistic theories of phonetic perception

Many neurolinguistic theories of phonetic perception have been proposed [23, 24, 25, 12, 26], but neurophysiological evidence to support them is limited. One broad class of models follows the paradigm of hierarchical processing first described by Hubel and Weisel in the visual system [23, 27, 24]. In these models, successive processing stages in the auditory system extract acoustic features with progressively increasing complexity by combining the simpler representations present in preceding stages. Such hierarchical processing is relatively well-supported by experimental data. For example, the responses of neurons in primary auditory cortex (A1) to speech sounds are more diverse than those in inferior colliculus [28] (but see [29]). While phoneme identity can be classified post-hoc from population-level activity in A1 [30, 31, 32], neurons in secondary auditory cortical regions explicitly encode higher-order properties of speech sounds [33, 34, 35, 36, 37].

Another class of models proposes that phonemes have no positive acoustic "prototype", and that we instead learn only the acoustic features useful for telling them apart [25]. Theoretically, these discriminative models provide better generalization and robustness to high variance [38]. Theories based on discrimination rather than prototype-matching have a long history in linguistics [39], but have rarely been implemented as neurolinguistic models. A possible neural implementation of discriminative perception is that informative contrast cues could evoke inhibition to suppress competing phonetic percepts, similar to predictive coding [40, 25, 41]. Neurophysiological evidence supports the existence of discriminative predictive coding, but its specific implementation is unclear [42, 43].

These two very different classes of models illustrate a major barrier faced by phonetic research: both classes can successfully predict human categorization performance, making it difficult to empirically validate or refute either of them using psychophysical experiments alone. Mechanistic differences have deep theoretical consequences — for example, the characterizations made by the above two classes of models re-

garding what phonemes *are* precisely oppose one another: are they positive acoustic prototypes, or sets of negative acoustic contrasts? Perceptually, do listeners identify phonemes, or discriminate between them? Neurobiological evidence regarding how the brain actually solves these categorization problems could help overcome this barrier.

The utility of a mouse model for speech research

Neurolinguistic research in humans faces several limitations that could be overcome using animal models.

First, most current human neurophysiological methods lack the spatiotemporal resolution to probe the fine spatial scale of neuronal circuitry and the millisecond timescale of speech sounds. A causal, mechanistic understanding of computation in neural circuits is also greatly aided by the ability to manipulate individual neurons or circuit components, which is difficult in humans. Optogenetic methods available in mice provide the ability to activate, inactivate, or record activity from specific types of neurons at the millisecond timescales of speech sounds.

Second, it is difficult to isolate the purely auditory component of speech perception in humans. Humans can use contextual information from syntax, semantics or task structure to infer phoneme identity [44, 45]. It is also difficult to rule out the contribution of multimodal information [46], or of motor simulation predicted by motor theories. Certainly, these and other non-auditory strategies are used during normal human speech perception. Nevertheless, speech perception is possible without these cues, so any neurocomputational theory of phonetic perception must be able to explain the purely auditory case. Animal models allow straightforward isolation of purely auditory phonetic categorization without interference from motor, semantic, syntactic, or other non-auditory cues.

Third, it is difficult to control for prior language experience in humans. Experience-dependent effects on phonetic perception are present from infancy [47]. It can therefore be challenging to separate experience-driven effects from innate neurocomputational constraints imposed by the auditory system. Completely language-naive subjects (such as animals) allow the precise control of language exposure, permitting phonetics and phonology to be disentangled in neurolinguistics.

Animal models of phonetic perception are a useful way to avoid these confounds, and provide an important alternative to human studies for empirically grounding the development of neurolinguistic theories. The mouse is particularly well-suited to serve as such a model. A growing toolbox of powerful electrophysiological and optogenetic methods in mice has allowed unprecedented precision in characterizing neural circuits and the computations they perform.

The utility of phonetics for auditory neuroscience

Conversely, auditory neuroscience stands to benefit from the framework provided by phonetics for studying how sound is transformed to meaning. Understanding how complex sounds are encoded and processed by the auditory system, ultimately leading to perception and behavior, remains a challenge for auditory neuroscience. For example, it has been difficult to extrapolate from simple frequency/amplitude receptive fields to understand the hierarchical organization of complex feature selec-

tivity across brain areas. A great strength of neuroethological model systems such as the songbird is that both the stimulus (e.g., the bird's own song) and the behavior (song perception and production) are well understood. This has led to significant advances in understanding the hierarchical organization and function of the song system [48, 49]. The long history of of speech research in humans has produced a deep understanding of the relationships between acoustic features and phonetic perception [50]. These insights have enabled specific predictions about what kinds of neuronal selectivity for features (and combinations of features) might underlie phonetic perception [1]. Although recognizing human speech sounds is not a natural ethological behavior for mice, phonetics nevertheless provides a valuable framework for studying how the brain encodes and transforms complex sounds into perception and behavior.

Here we trained mice to discriminate between pitch-shifted recordings of naturally produced consonant-vowel (CV) pairs beginning with either /g/ or /b/. Mice demonstrated the ability to generalize consonant identity across novel vowel contexts and speakers, consistent with true category learning. To our knowledge this is the first demonstration that any animal can generalize consonant identity across both novel vowel contexts and novel speakers. These results indicate that mice can solve the

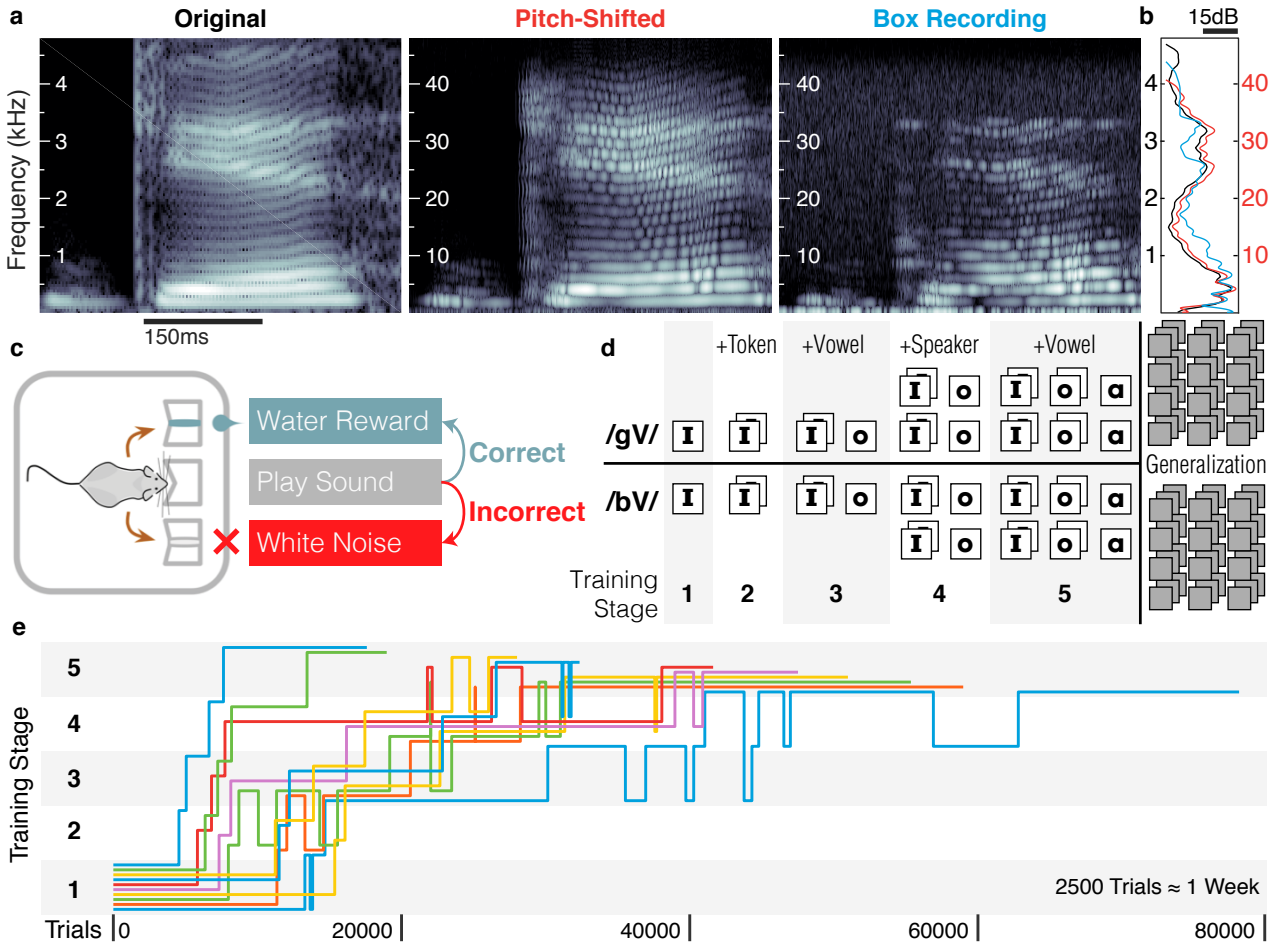


Figure 1.1: Stimuli and Task Design. a) Spectrograms of stimuli. Left: Example of an original recording of an isolated consonant-vowel token (/gI/). Center: the same token pitch-shifted upwards by 10x (3.3 octaves) into the mouse hearing range. Right: Recording of the pitch-shifted token presented in the behavior box. Stimuli retained their overall acoustic structure

non-invariance problem, and suggest that mice are a suitable model for studying phonetic perception.

1.2 Results

Generalization performance

We began training 23 mice to discriminate between consonant-vowel (CV) pairs beginning with either /b/ or /g/ in a two-alternative forced choice task. CV tokens were pitched-shifted up into the mouse hearing range (Fig. 1.1a-b). Each mouse began training with a pair of tokens (individual recordings) in a single vowel context (ie. /bI/ and /gI/) from a single speaker, and then advanced through stages that progressively introduced new tokens, vowels, and speakers (Fig. 1.1c-d, see Methods). Training was discontinued in 13 (56.5%) of these mice because their performance on the first stage was not significantly better than chance after two months. The remaining 10 (43.5%) mice progressed through all the training stages to reach a final generalization task, on average in 14.9 $(\sigma \pm 7.8)$ weeks (Fig. 1.1e). This success rate and training duration suggest that the task is difficult but achievable.

We note that this training time is similar to that reported previously for rats $(14 \pm 0.3$weeks [18]). Previous studies have not generally reported success rates. Human infants also vary in the rate and accuracy of their acquisition of phonetic categories [**?**], so we did not expect perfect accuracy from every mouse. The cause of such differences in ability is itself an opportunity for future study.

Generalization is an essential feature of categorical perception. By testing whether mice can generalize their phonetic categorization to novel stimuli, we can distinguish whether mice actually learn phonetic categories or instead just memorize the reward contingency for each training token. Four types of novelty are possible with our stimuli: new tokens from the speakers and vowel contexts used in the training set, new vowels, new speakers, and new vowels from new speakers (colored groups in Fig. 1.2a). In the final generalization stage, we randomly interleaved tokens from each of these novelty classes on 20% of trials, with the remaining 80% consisting of tokens from the training set. We interleaved novel tokens with training tokens for two reasons: (1) to avoid a sudden increase in task difficulty, which can degrade performance, and (2) to minimize the possibility that mice could learn each new token by widely separating them in time (on average, generalization tokens were repeated only once every five days).

We looked for 4 hallmarks of generalization: (1) Mice should be able to accurately categorize novel tokens, (2) performance should reflect the quality of the acoustic-phonetic criteria learned in training, (3) performance on novel tokens should be correspondingly worse for tokens that differ more from those in the training set, and (4) accurate categorization of novel tokens should not require additional reinforcement.

All 10 mice were able to categorize tokens of all generalization types with an accuracy significantly greater than chance. We estimated the impact of each generalization class on performance as a fixed factor nested within each mouse as a random factor in a mixed-effects logistic regression (see Methods). The predicted accuracy for each generalization class is shown in Table 1.1, each providing an estimate of the difficulty of that class after accounting for the random effects of individual mice.

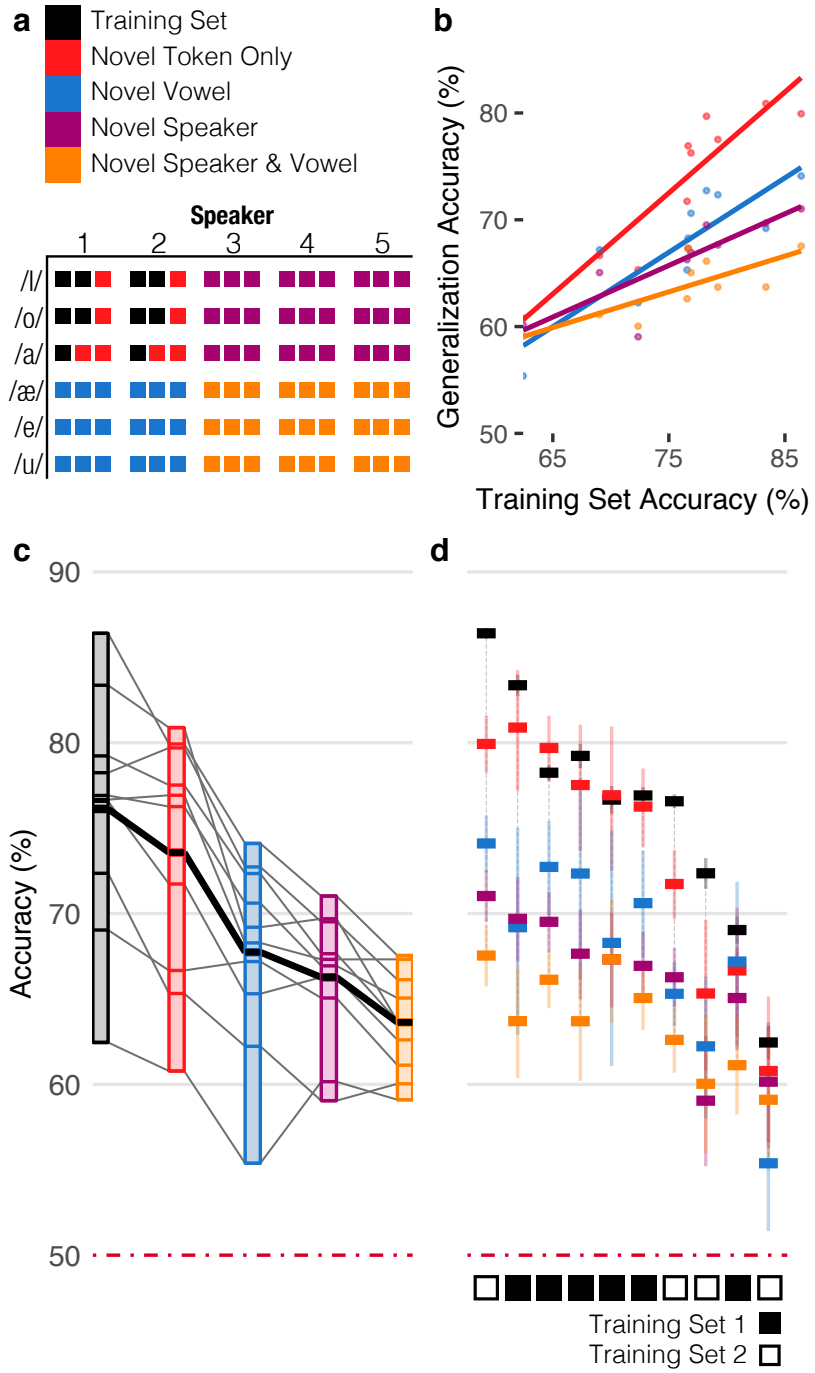


Figure 1.2: Generalization accuracy by novelty class. Mice generalized stop consonant discrimination to novel CV recordings. **a)** Four types of novelty are possible with our stimuli: novel tokens from the speakers and vowels used in the training set (red), novel vowels (blue), novel speakers (purple), and novel speakers with novel vowels (orange). Tokens in the training set are indicated in black. Colors same throughout. **b)** Mice that performed better on the training set were better at generalization. Each point shows the performance for a single mouse on a given novelty class, plotted against that mouse's performance on training tokens presented on during the generalization phase (both averaged across the entire generalization phase). Lines show linear regression for each novelty class. **c)** Mean accuracy for each novelty class (gray lines indicate individual mice). **d)** Mean accuracy for individual mice (colored bars indicate each novelty class). Error bars in **d** are 95% binomial confidence intervals. Mice were assigned one of two sets of training tokens, black and white boxes in **d**.

Performance on all generalization types was strongly and positively correlated with performance on the training set (Fig. 1.2b, adj. $R^2 = 0.74, F(4,5) = 7.4, p < 0.05$). If mice were "overfitting," that is, memorizing the training tokens rather than learning categories, then we would expect the opposite (i.e., above some threshold, mice that performed better on the training set would perform correspondingly worse on the generalization set). It appears instead that better prototypes or deci-

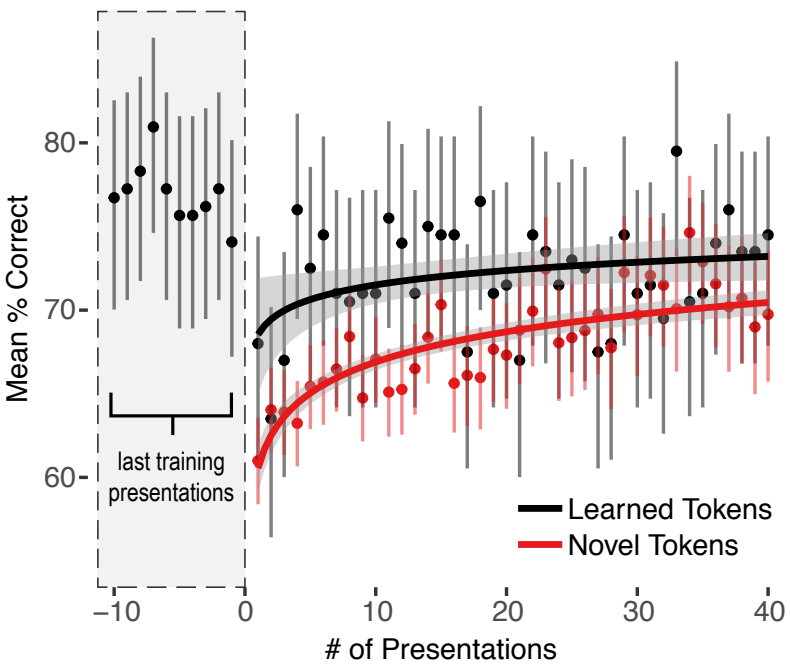


Figure 1.3: Learning curve for novel tokens. Performance for both novel and training set tokens dropped transiently and recovered similarly after the transition to the generalization stage. Presentation 0 corresponds to the transition to the generalization stage. The final ten trials before the transition are shown in the gray dashed box. Mean accuracy and 95% binomial confidence intervals are collapsed across mice for novel (red, all novelty classes combined) or learned (black) tokens, by number of presentations in the generalization task. Logistic regression of binomial correct/incorrect responses fit to log-transformed presentation number (lines, shading is smoothed standard error).

sion boundaries learned in the training stages allowed better generalization to novel tokens.

Mice were better at some types of generalization than others (Fig. 1.2c). The estimates of their relative difficulty (Fig. 1.2c) provide a ranking of the perceptual novelty of the stimulus classes based on their similarity to the training tokens. From easiest to hardest, these were: novel token, novel vowel, novel speaker (which was not significantly more difficult than novel vowel), novel speaker & vowel. The effects of generalizing to novel vowels and novel speakers were not significantly different from each other, but pairwise comparisons between each of the other types of generalization were (Tukey's method, all $p < 0.001$, also see confidence intervals in Table 1.1).

Although the effect of each generalization type on performance was significantly different between mice (Likelihood Ratio Test, $\chi^2(14) = 407.22, p \ll 0.001$), they were highly correlated (see Table 1.1). The relative consistency of novelty type difficulty across mice (ie. the correlation of fixed effects, Fig. 1.2c) is striking, but our results cannot distinguish whether it is due to the mice or the stimuli: it is unclear whether the acoustic/phonetic criteria learned by all mice are similarly general, or whether the "cost" of each type of generalization is similar across an array of possible acoustic/phonetic criteria.

True generalization requires that one set of discrimination criteria can be successfully applied to novel cases without reinforcement. It is possible that the mice were instead able to rapidly learn the reward contingency of novel tokens during the generalization stage. If mice were learning rapidly rather than generalizing, this would predict that novel token performance (1) would be indistinguishable from chance on the first presentation, and (2) would increase relative to performance on already-learned tokens with repeated presentations.

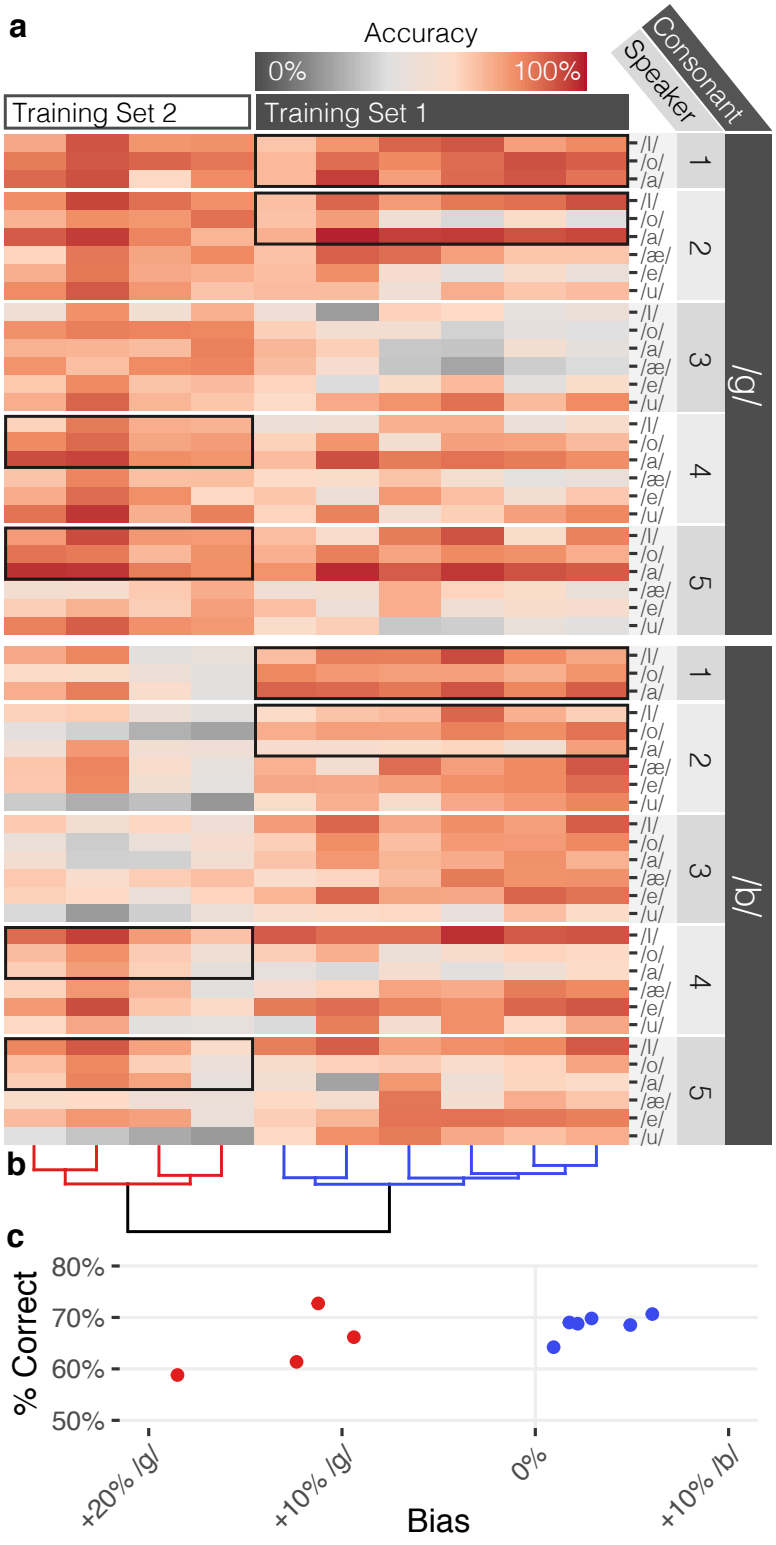


Figure 1.4: Patterns of individual and group variation. a) Mean accuracy (color, scale at top) for each mouse (columns) on tokens grouped by consonant, speaker, and vowel (rows). The different training sets (cells outlined with black boxes) led to different patterns of accuracy on the generalization set. **b)** Ward clustering dendrogram, colored by cluster. **c)** Training set cohorts differed in bias but not mean accuracy.

Performance on the first presentation of novel tokens was significantly greater than chance (Fig. 3, all mice, all tokens from all novelty classes: one-sided binomial test, $n = 1410$, $P_{correct} = 0.61$, lower 95% CI $= 0.588$, $p \ll 0.001$; all mice, worst

novelty class: $n = 458, P_{correct} = 0.581$, lower 95% CI $= 0.541, p < 0.001$). This demonstrates that mice were able to generalize immediately without additional reinforcement. Although performance on novel tokens did increase with repetition, so did performance on training tokens (Fig. 1.3). We noted that performance on all tokens (both novel and previously learned tokens) transiently dropped after each transition between task stages, suggesting a non-specific effect of an increase in task difficulty. To distinguish an increase in performance due to learning from an increase due to acclimating to a change in the task, we compared performance on generalization and training tokens over the first 40 presentations of each token. If the mice were learning the generalization tokens, the increase in performance with repeated presentations should be significantly greater than that of the already trained tokens.

Performance was well fit by a logistic regression of correct/incorrect responses from each mouse against the novelty of a token (trained vs. novel tokens), and the number of times it had been presented (Fig. 1.3). The effect of the number of presentations on accuracy was not significantly different for novel tokens compared to trained tokens (interaction between novelty and the number of presentations: Wald test, $z = 1.239, 95\% CI = [-0.022, 0.1], p = 0.215$). This was also true when the model was fit with with the generalization types themselves rather than trained vs. novel tokens (most significant interaction, generalization to novel speakers x number of presentations: Wald test, $z = 1.425, 95\% CI = [-0.018, 0.117], p = 0.154$) and with different numbers of repetitions (10: $z = -0.219, 95\% CI = [-0.161, 0.13], p = 0.827$; 20: $z = -0.521, 95\% CI = [-0.116, 0.068], p = 0.602$). This indicates that the asymptotic increase in performance on novel tokens was a general effect of adapting to a change in the task rather than a learning period for the novel stimuli.

In summary, the behavior of the mice is consistent with an ability to generalize some learned acoustic criteria to novel stimuli. It is unlikely that the mice rapidly learned the novel tokens because (1) performance on the first presentation of novel tokens was significantly above chance, (2) performance on subsequent presentations of novel tokens did not improve compared to trained tokens, and (3) learning each token would have to take place over unrealistically long timescales: there were an average of 2355 trials (5 days) between the first and second presentation of each novel token.

Training Set Differences

One strength of studying phonetic perception in animal models is the ability to precisely control exposure to speech sounds. To test whether and how the training history impacted the pattern of generalization, we divided mice into two cohorts trained with different sets of speech tokens. In the first cohort ($n = 6$ mice), mice were trained with tokens from speakers 1 and 2 (speaker number in Fig. 1.4a), whereas the second cohort ($n = 4$ mice) were trained on speakers 4 and 5.

The two training cohorts had significantly different patterns of which tokens were accurately categorized (Fig. 1.4a, Likelihood-Ratio test, regression of mean accuracy on tokens with and without token x cohort interaction: $\chi^2_{161}, p \ll 0.001$). Put another way, accuracy patterns were markedly similar within training cohorts: cohort differences accounted for fully 40.6% of all accuracy variance (sum of squared-error) between tokens.

Mice from the second training cohort were far more likely to report novel tokens as a /g/ than the first cohort (Fig. 1.4b), an effect that was not significantly related to their overall accuracy ($b = 0.351, t(8) = 2.169, p = 0.062$). Since the only difference between these mice were the tokens they were exposed to during training (they were trained contemporaneously in the same boxes), we interpret this response bias as the influence of the training tokens on whatever acoustic cues the mice had learned in order to perform the generalization task. This suggests that the acoustic properties of training set 2 caused the /g/ "prototype" to be overbroad.

We searched for additional sub-cohort structure with hierarchical clustering (Ward's Method, dendrogram in fig 1.4b). Within each training cohort there appeared to be two additional clusters of accuracy patterns. Though our sample size was too small to meaningfully interpret these clusters, they raise the possibility that even when trained using the same set of sounds mice might learn multiple sets of rules to distinguish between consonant classes.

Acoustic-behavioral correlates

Humans can flexibly use several acoustic features such as burst spectra and formant transitions to discriminate plosive consonants, and we wondered to what extent mice were sensitive to these same features.

One dominant acoustic cue for place of articulation in stop consonants is the transition of the second formant following the plosive burst [**?** 1, 51]. Formant transitions are complex and dependent on vowel context, but tokens for a given place of articulation cluster around a line – or "locus equation" – relating F2 frequency at release to its mid-vowel steady-state [1, 51] (Fig. 1.5a). If mice were sensitive to this cue, the distance from both locus equation lines should influence response. For example, a /g/ token between the locus equation lines should have a greater rate of misclassification than a token at an equal distance above the red /g/ line. Therefore we tested how classification depended on the difference of distances from each line (/g/ distance - /b/ distance, which we refer to as "locus difference").

Mean responses to tokens (ranging from 100% /g/ - 100% /b/) were correlated with locus differences (black line, Fig. 1.5b). However, it is important to note that this correlation does not necessarily demonstrate that mice relied on this acoustic cue. Because multiple acoustic features are correlated with consonant identity, performance that is correlated with one such cue would also be correlated with all the others. The mice learned some acoustic property of the consonant classes, and since the acoustic features are all highly correlated with one another, they are all likely to correlate with mean responses.

To distinguish whether mice specifically relied on F2 locus distance, we therefore measured the marginal effect of this acoustic cue within a consonant class. This is shown by the slopes of the red and blue lines in Fig. 1.5b. For example, is a /g/ token that is further away from the blue /b/ line more likely to be identified as a /g/ than one very near the /b/ line? Mean responses to /g/ tokens were negatively correlated with locus distance (Mean response /g/ to /b/ between 0 and 1, $b = -0.028kHz, 95\%CI = [-0.035, -0.022], p \ll 0.001$). In other words, tokens that should have been more frequently confused with /b/ were actually more likely to be classified as /g/. Note the red points at locus distance of zero in Fig. 1.5b: these tokens have an equal distance from both the /b/ and /g/ locus equa-

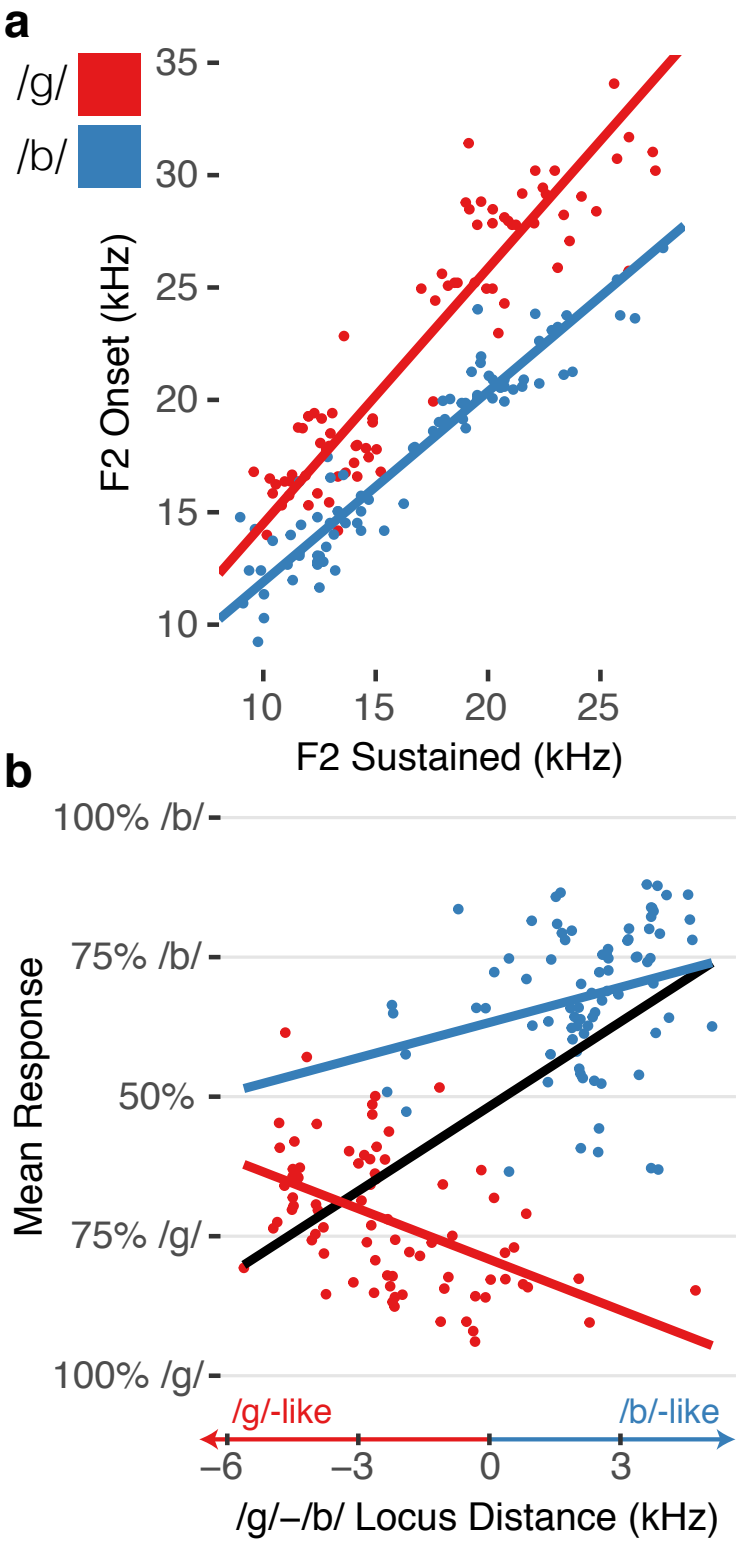


Figure 1.5: Acoustic-Behavior Correlates F2 Onset-Vowel transitions do not explain observed response patterns. **a)** Locus equations relating F2 at burst onset and vowel steady state (sustained) for each token (points), split by consonant (colors, same as **b)**). **b)** As the difference of a token's distance from the ideal /g/ and /b/ locus equation lines increased (x axis, greater distance from /g/, smaller distance from /b/), /b/ tokens obeyed the predicted categorization while /g/ tokens did not (slopes of colored lines).

tion prototypes but are some of the most accurately categorized /g/ tokens. /b/ tokens obeyed the predicted direction of locus distance ($b = 0.049, 95\% CI =$

$[0.039, 0.06], p \ll 0.001$), but the effect was very small: moving one standard deviation ($\sigma_{/b/} = 1.618kHz$) towards the /g/ line only changed responses by 7.9%. These results suggest that mice did not rely on F2 transitions to categorize these consonants.

We repeated this analysis separately for each training cohort to test whether the two cohorts could have developed different acoustic templates that better explained their response patterns. We derived cohort-specific locus-equation lines and distances using only the tokens from each of their respective training sets. These models were qualitatively similar to the model that included all tokens and mice and did not improve the model fit (Cohort 1: /g/: $b = -0.051, 95\%CI = [-0.064, -0.038]$, /b/: $b = 0.041, 95\%CI = [0.022, 0.059]$; Cohort 2: /g/: $b = -0.022, 95\%CI = [-0.031, -0.014]$, /b/: $b = 0.055, 95\%CI = [0.042, 0.069]$).

We conclude that while our stimulus set had the expected F2 formant transition structure, this was unable to explain the behavioral responses we observed both globally and within training cohorts. There are, of course, many more possible acoustic parameterizations to test, but the failure of F2 transitions to explain our behavioral data is notable because of its perceptual dominance in humans and its common use in parametrically synthesized speech sounds. This demonstrates one advantage of using natural speech sounds: mice trained on synthesized speech that varied parametrically only on F2 transitions would likely show sensitivity to this cue, but this does not mean that mice show the same feature sensitivity when trained with natural speech. Preserving the complexity of natural speech stimuli is important for developing a general understanding of auditory category learning.

1.3 Discussion

These results demonstrate that mice are capable of learning and generalizing phonetic categories. Indeed, this is the first time to our knowledge that mice have been trained to discriminate between any classes of natural, non-species-specific sounds. Thus mice join a number of model organisms that have demonstrated categorical learning with speech sounds [6**?** , 21, 17, 18, 19, 20], making a new suite of genetic and electrophysiological tools available for phonetic research.

Two subgroups of our mice that were trained using different sets of speech tokens demonstrated distinct patterns of consonant identification, presumably reflecting differences in underlying acoustic prototypes. The ability to precisely control exposure to speech sounds provides an opportunity to probe the neurocomputational constraints that govern the possible solutions to consonant identification.

Here we opted to use naturally recorded speech tokens in order to demonstrate that mice could perform a "hard version" of phonetic categorization that preserves the full complexity of the speech sounds and avoids *a priori* assumptions about the parameterization of phonetic contrasts. Although our speech stimuli had the expected F2 formant transition structure, that did not explain the response patterns of our mice. This suggests that the acoustic rules that mice learned are different from those that would be learned from synthesized speech varying only along specifically chosen parameters.

Future experiments using parametrically synthesized speech sounds are a critical next step, and will support a qualitatively different set of inferences. Being able to carefully manipulate reduced speech sounds is useful to probe the acoustic cue

structure of learned phonetic categories, but the reduction in complexity that makes them useful also makes it correspondingly more difficult to probe the learning and recognition mechanisms for a perceptual category that is defined by multiple imperfect, redundant cues. It is possible that the complexity of natural speech may have caused our attrition rate to be higher, and task performance lower, than other sensory-driven tasks. Neither of those concerns, however, detracts from the possibility for the mouse to shed mechanistic insight on phonetic perception. Indeed, error trials may provide useful neurophysiological data about how and why the auditory system fails to learn or perceive phonetic categories.

We hope in future experiments to directly test predictions made by neurolinguistic models regarding phonetic acquisition and discrimination. For example, one notable model proposes that consonant perception relies on combination-sensitive neurons that selectively respond to specific combinations of acoustic features [1]. This model predicts that mice trained to discriminate stop consonants would have neurons selective for the feature combinations that drive phoneme discrimination, perhaps in primary or higher auditory cortical areas. Combination-selective neurons have been observed in A1 [52, 53], and speech training can alter the response properties of A1 neurons in rats [18], but it is unclear whether speech training induces combination-selectivity that would facilitate phonetic discrimination.

The ability to record from hundreds of neurons in awake behaving animals using tetrode electrophysiology or 2-photon calcium imaging presents exciting opportunities to test predictions like these. Should some candidate population of cells be found with phonetic selectivity, the ability to optogenetically activate or inactivate specific classes of neurons (such as excitatory or inhibitory cell types, or specific projections from one region to another) could shed light on the circuit computations and transformations that confer that selectivity.

1.4 Methods

Animals

All procedures were performed in accordance with National Institutes of Health guidelines, as approved by the University of Oregon Institutional Animal Care and Use Comittee.

We began training 23 C57BL/6J mice to discriminate and generalize stop consonants in CV (consonant-vowel) pairs. 13 mice failed to learn the task (see Training, below). 10 mice (43.5%) progressed through all training stages and reached the generalization task in an average 14.9 ($\sigma = 7.8$) weeks. Mean age at training onset was 8.1 ($\sigma = 2$) weeks, and at discontinuation of training was 50.6 ($\sigma = 11.2$) weeks. Sex did not significantly affect the probability of passing or failing training (Fisher's Exact Test: $p = 0.102$), nor did the particular behavioral chamber used for training ($p = 0.685$), nor age at the start of training (Logistic regression: $z = 1.071, p = 0.284$). Although this task was difficult, our training time (14 ± 0.3 weeks as in [18]), and accuracy (generalization: 76%[6], training tokens only: 84.1% [18]) are similar to comparable experiments in other animals.

Speech stimuli

Speech stimuli were recorded in a sound-attenuating booth with a head-mounted microphone attached to a Tascam DR-100mkII handheld recorder sampling at 96kHz/24bit. Each speaker produced a set of 3 recordings (tokens) of each of 12 CV pairs beginning with either /b/ or /g/, and ending with /I/, /o/, /a/, /æ/, /ɛ/, /u/. To reduce a slight hiss that was present in the recordings, they were denoised using a Daubechies wavelet with two vanishing moments in MATLAB. The typical human hearing range is 20 Hz - 20 kHz, whereas the mouse hearing range is 1 kHz - 80 kHz [54]. The F_0 of our recorded speech sounds ranged from 100 - 200 Hz, which is well below the lower frequency limit of the mouse hearing range. We therefore pitch shifted all stimuli upwards by 10x (3.3 octaves) in MATLAB [**?**]. This shifted all spectral information equally upwards into an analogous part of mouse hearing range while preserving temporal information unaltered.

Tokens from five speakers (one male - speaker 1 throughout, four female - speakers 2-5 throughout) were used. Three vowel contexts (/æ/, /ɛ/, and /u/) were not recorded from one speaker. It is unlikely that this had any effect on our results, as our primary claims are based on the ability to generalize at all, rather than generalization to tokens from a particular speaker. Tokens were normalized to a common mean amplitude, but were otherwise unaltered to preserve natural variation between speakers — indeed, preserving such variation was the reason for using naturally recorded rather than synthesized speech.

Formant frequency values were measured manually using Praat [55]. F2 at onset was measured at its center as soon as it was discernible, typically within 20 ms of burst onset, and at vowel steady-state, typically 150-200ms after burst onset.

Training

We trained mice to discriminate between CV pairs beginning with /b/ or /g/ in a two-alternative forced choice task. Training sessions lasted approximately 1 hour, 5 days a week. Each custom-built sound-attenuating training chamber contained two free-field JBL Duet speakers for stimulus presentation with a high-frequency rolloff of 34 kHz, and a smaller 15 x 30 cm plastic box with three "lick ports." Each lick port consisted of a water delivery tube and an IR beam-break sensor mounted above the tube. Beam breaks triggered water delivery by actuating a solenoid valve. Water-restricted mice were trained to initiate each trial with an unrewarded lick at the center port, which started playback of a randomly selected stimulus, and then to indicate their stimulus classification by licking at one of the ports on either side. Tokens beginning with /g/ were always on the left, with /b/ on the right. Two cohorts were trained on two separate sets of tokens. Training set 1 started with speaker 1 (Fig. 4a) and had speaker 2 introduced on the fourth stage, where Training set 2 started training with speaker 5 and had speaker 4 introduced on the fourth stage. Correct classifications received ∼10 µL water rewards, and incorrect classifications received a 5s time-out that included a mildly aversive 60 dB SPL white noise burst.

Training advanced in stages that progressively increased the number of tokens, vowel contexts, and speakers. Mice first learned a simple pure-tone frequency discrimination task to familiarize them with the task and shape their behavior; the tones were gradually replaced with the two CV tokens of the first training stage. CV discrimination training proceeded in 5 stages outlined in Table 2. Mice automatically gradu-

ated from each stage when 75% of the preceding 300 trials were answered correctly. In a few cases, a mouse was returned to the previous stage if its performance fell to chance for more than a week after graduating. Training was discontinued after two to three months if performance in the first stage never rose above chance. Mice that reached the final training stage were allowed to reach asymptotic performance, and then advanced to a generalization task.

In the generalization task, stimuli from the set of all possible speakers, vowel contexts, and tokens (140 total, not including the stage 5 stimulus set) were randomly presented on 20% of trials and the stage 5 stimulus set was used on the remaining 80%. Training tokens were drawn from a uniform random distribution so that each was equally likely to occur during both the stage 5 training and generalization phases. Novel tokens were drawn uniformly at random by their generalization class, but since there were unequal numbers of tokens in each class (Novel token only: 16 tokens, Novel Vowel: 36, Novel Speaker: 54, Novel Speaker + Vowel: 54), tokens in each class had an unequal number of presentations. We note that the logistic regression analysis with restricted maximum likelihood that we used is robust to unequal sample sizes [56].

Data analysis

Data were excluded from days on which a mouse had a $>$ 10% drop in accuracy from their mean performance on the previous day ($44/636 = 7\%$ of sessions). Anecdotally, mice are sensitive to environmental conditions (e.g., thunderstorms), so even though all efforts were made to minimize variation between days, even the best performing mice had "bad days" where they temporarily fell to near-chance performance and exhibited strong response bias. We thus assume these "bad days" were the result of temporary environmental or other performance issues, and were unrelated to the difficulty of the task itself.

All analyses were performed in R (R version 3.5.3 (2019-03-11))[57] using RStudio (1.1.456)[58]. Generalization performance was modeled using a logistic generalized linear mixed model (GLMM) using the R package "lme4"[59]. Binary correct/incorrect responses were fit hierarchically to models of increasing complexity (see Table 1.3), with a final model consisting of the generalization class (as in Fig. 2a: training tokens, novel tokens from the speakers and vowels in the training set, novel speaker, novel vowel, and novel speaker and vowel) as a fixed effect with random slopes and intercepts nested within each mouse as a random effect. There was no evidence of overdispersion (i.e., deviance $\approx$ degrees of freedom, or less than ~ 2 times degrees of freedom), and the profile of the model showed that the deviances by each fixed effect were approximately normal. Accordingly, we report Wald confidence intervals. We also computed bootstrapped confidence intervals, which had only minor disagreement with the Wald confidence intervals and agreed with our interpretation in the text.

Clustering was performed with the "cluster"[60] package. Ward clustering split the mice into two notable clusters, which are plotted in Fig. 1.4.

We estimated locus equations relating F2 onset and F2 vowel using total least squares linear regression. The locus equations of the /b/ and /g/ tokens accounted for 97.3% and 95.9% of the variance in the F2 measurements of our tokens, respectively.

Spectrograms in Figure 1.1a were computed with the "spectrogram" function in

MATLAB 2017b, and power spectra in Figure 1.1b were computed with the "pwelch" function in MATLAB 2018b with the same window and overlap as 1.1a spectrograms.

The remaining analyses are described in the text and used the "binom"[61], "reshape"[**?**], and "plyr"[62] packages. Data visualization and tabulation was performed with the "ggplot2,"[63] and "xtable"[64] packages.

1.5 Tables

	Accuracy	95% Wald CI	Corr			
Learned	0.767*	[0.748, 0.785]				
Token	0.739*	[0.713, 0.763]	0.5			
Vowel	0.678*	[0.655, 0.701]	0.81	0.91		
Speaker	0.666*	[0.651, 0.68]	0.98	0.68	0.92	
Vow+Spk	0.637*	[0.624, 0.651]	0.98	0.64	0.9	1

Table 1.1: Impact of each generalization class on performance. Accuracy values provide an estimate of the difficulty of that class after accounting for the random effects of individual mice. Accuracies are logistic GLMM coefficients transformed from logits, and model coefficients are logit differences from training set accuracy, which was used as an intercept. Correlation values are between fixed effects (novelty classes) across random effects (mice). * indicates significance ($p(>|z|) \ll .001$).

Stage	Speakers	Vowels	Total Tokens
1	1	1	2
2	1	1	4
3	1	2	6
4	2	2	12
5	2	3	20
Generalization	5	6	160 (20 training, 140 novel)

Table 1.2: Token structure of training stages

	DF	χ^2	DF_{χ^2}	$\Pr(>\chi^2)$
Mouse	2			
Mouse + Type	6	2534.46	4	$\ll 0.001$
Type | Mouse	20	407.22	14	$\ll 0.001$

Table 1.3: hierarchical GLMM: To reach the appropriate complexity of model, we first modeled correct/incorrect answers as a function of each mouse as a fixed effect (row 1), then added the generalization type (as in Fig. 1.2) as a fixed effect (row 2), and finally modeled generalization type as a fixed effect nested within each mouse as a random effect (row 3). Since the final model had the best fit, it was used in all reported analyses related to the GLMM.

2

Speech Modeling

2.1 Abstract

[abstract itself]

[summary of intellecual merit, broader impacts?]

- why haven't we done these experiments already? what's the role of animal models? what's the way forward??? neurophys in animals as speech models, but what *kind* of experiments are likely to help us with this question of what phonemes are.

2.2 Introduction

Phonemes are Language Games

> "Consider for example the proceedings that we call "games". [...] For if you look at them you will not see something that is common to all, but similarities, relationships, and a whole series of them at that. [...] Are they all 'amusing'? Compare chess with noughts and crosses. Or is there always winning and losing, or competition between players? Think of patience. [...] Look at the parts played by skill and luck; and at the difference between skill in chess and skill in tennis.
>
> And the result of this examination is: we see a complicated network of similarities overlapping and criss-crossing: sometimes overall similarities, sometimes similarities of detail. [...] And we extend our concept as in spinning a thread we twist fibre on fibre. And the strength of the thread does not reside in the fact that some one fibre runs through its whole length, but in the overlapping of many fibres."
>
> *-Wittgenstein, Philosophical Investigations: 66-67[65]*

Cognitive reality is characterized by its discreteness: rather than a continuous undifferentiated gradient wash of sensation and cognition, we experience objects, concepts, and categories. Speech is a continuous, high-dimensional, high-variability acoustic signal, yet it is perceived as a small number of relatively-discrete phonemes[**?**]. The acoustic structure of phonemes is a sort of "Family Resemblance"[65] — the truly extravagant variability of speech has thus far defied any simple, definite acoustic parameterization of its phonemes. Instead, individual utterances within a phonetic category vary along high numbers of feature-dimensions, none of which are necessary nor sufficient for a listener to identify it[66, 67].

There are different types of category structure, and what typifies family resemblance structures is 1) multiply defined - category membership is assesed across many imperfect 'features' none of which is necessary nor sufficient, 2) prototypicality - some instances are better 'examples' of a category than others, category membership is not bi-

nary, 3) context dependent - which feature is important depends on the features present in the instance and the context in which it is being compared. [68]

A Very Simple Model...

Category representation theories are intimately related (and occasionally literally isometric to [69]) to theories of the measurement of similarity, which is dominated by geometric models[70]. These models nearly universally presuppose that categories exist in a feature space such that there exist some number of features that describe each instance of an object to be categorized.

To begin perhaps purposely naively, we will formulate a very simple geometric model of perceptual categories:

Suppose that some sensory stimulus **s** was composed of some set of physical attributes a_i in the d-dimensional "stimulus space" **S** capable of fully representing all stimuli for a given sensory modality (as opposed to a particular set of eg. parameterized stimuli)

$$\mathbf{s} = \{a_0, a_i, \dots a_d \ : a \in \mathbf{S}\} \tag{2.1}$$

For example, a digital sound is fully defined by the amplitudes of the waveform at each of its samples, or an image is defined as the wavelength and intensity of light at each pixel. Since a_i are arbitrary, **S** can represent a set of static attributes, or a set of attributes through time.

The sensory stimulus **s** is processed into some percept **p** composed of perceptual attributes b_i in the e-dimensional "perceptual space" **P**

$$\mathbf{p} = \{b_0, b_i, \dots b_e \ : b \in \mathbf{P}\} \tag{2.2}$$

such that some perceptual computation M maps **S** to **P**.

$$M = f : \mathbf{S} \to \mathbf{P} \tag{2.3}$$

$$\mathbf{p} = M(\mathbf{s}) \tag{2.4}$$

Like **S**, the form of **P** is arbitrary, so while the discussion that follows treats it like a continuously-valued metric space, it could also consist of a collection of binary/discrete properties (like traditional phonetic descriptions like [± voiced]), as in, for example [70, 71]

The objective of the observer is to infer the category c_s given **s**'s representation as **p**.

$$c_s = max(\{p(c_i|\mathbf{p}) : c_i \in \mathbf{C}\}) \tag{2.5}$$

The form of the sensory-perceptual mapping M, the perceptual space **P** it constructs, and the inference of category identity c_s it supports serve as a loom for a few threads of the speech perception problem scattered across a few disciplines and vocabularies.

...and its history

Make sure to refer back to the 3 properties of family resemblance categories and use that to structure this section!!!

A prominent strain of phonetics research in the US, largely associated with the Haskins Labs ([72] and see [73, p. 51]), has characterized the speech perception problem as resolving a set of acoustic "cues" into phonetic identity:

> "Liberman, Cooper, and Pierre Delattre began to study the acoustic speech signal, to determine how it represents the consonants and vowels of spoken words, and to discover the acoustic structure (the 'cues') essential for their identification by listeners. [...] By selectively including and eliminating elements of acoustic structure,l Liberman and his colleagues could determine what bits of structure provided information for the different phonetic properties of spoken words."
>
> -Carol Fowler & Katherine S. Harris in [73, p. 51]

The "cue discovery" paradigm of phonetics research posits that, for the auditory component of phonetic perception, the elements in **P** are linear combinations of the features in **S** whose manipulation can influence the identity of the perceived phoneme. These features represent familiar phonetic parameterizations like voice onset times or formant frequency ratios. The mapping M that constructs **p** is taken to be a fixed, innate feature of the auditory system: "this version of the auditory theory takes the perceived boundary between one phonetic category and another to correspond to a naturally-occurring discontinuity in perception of the relevant acoustic continuum." [74].

The conclusion of cue-based research is summarized neatly by Philip, Robert E. Remez, and Jennifer Pardo with respect to their sinewave synthesis experiments: "Question: Which acoustic elements are essential for the perception of speech? Answer: None[75]." The failure to find a simple parameterization of phonetic categories as acoustic cues motivated an abandonment of an acoustic account of phonetic perception entirely in favor of a motor theory of perception that positied a special, evolved "speech module" that linked the wily acoustics of speech sounds to the action of the articulatory system:

> "For if phonetic categories were acoustic patterns, and if, accordingly, phonetic perception were properly auditory, one should be able to describe quite straightforwardly the acoustic basis for the phonetic category and its associated percept. According to the motor theory, by contrast, one would expect the acoustic signal to serve only as a source of information about the gestures; hence the gestures would properly define the category" [74]

Purely motor theories of speech have been diversely problematized, not least of all by the many demonstrations that animals that conspicuously lack a human articulatory system are capable of phonetic categorization[13**?** , 21]. The acoustic problem of speech perception was simply too difficult to be solved by an evolutionarily plausible auditory system – how could the family resemblance structure of phonetic categories be learned without some explicit, innate knowledge of the acoustic consequences of articulation?[67]

Research on infant acquisition of speech sounds has since demonstrated the profound plasticity of the auditory system and its ability to learn the complex statisti-

cal dependencies between the acoustic attributes of speech[76]. A family of models based primarily on the work of Patricia Kuhl and colleagues describe the stimulus space **S** as acoustic features based on the "basic cuts" of sensitivity in the auditory system[77]. Infants exploit the statistical regularity and patterns of feature co-occurance to learn some mapping M that constructs a "warped" perceptual space **P** that clusters features in **S** into acoustic "prototypes."[76]

Phonetic category identity then consists of some density in **P**, the center of which is the "ideal" phonetic exemplar most likely to be identified with a particular categoery, and proceeding from this center point one transitions from off-target imperfect examplars to overlapping densities of other phonetic categories. Extensions to the model make this formulation explicit, like Kronrod, Coppess, and Feldman's[3] bayesian model that offers a unified explanation of the strong categorical perception of stop consonants and the weaker categorical perception of vowels. Their model describes phonetic identification as an inference problem that depends on both the acoustic properties of a stimulus and prior knowledge of phonetic categories, defined as some mean and variance in an arbitrary perceptual space.

In this model, the difficulty of the acoustic problem of speech perception carefully described by cue-centric phonetic research is resolved by suggesting the auditory system relies on sharp internal representations of category identity for phonemes that have a large degree of uninformative variaance, like stop consonants.

The degree of arbitrariness is problematic for the model, however. The proposition that there is some stimulus space **P** *that supports linearly-separable phonetic categories is emphatically counterevidenced by the 70 years of cue-based research that has attempted to find one (cite violations of gestalt principles from [78] and [66]). These prototype models, without weighting for the informativeness of a particular dimension in context (as opposed to some global weight) would be vulnerable to misidentifying speech when the most dominant cue was made redundant, when in fact human listeners will adapt to using a more informative cue. In fact a lot of the research relies on carefully parameterized speech, so if they considered the cases where those cues failed then such a single-density-based prototype model. Having nonlinear blobby parameterizations of prototypes doesn't really solve the problem either, as you would then just require an additional downstream 'readout' layer that could compute the conditions where a particular dimension*

Their future directions says that identifying and learning the dimensions is of critical importance, we can extend our model by continuing Kronrod's emphasis on the information contained in each perceptual dimension and allow it to vary by context...

An extension to our model...

Instead of a static perceptual space **P** where a given stimulus **s** is mapped to a single percept **p** (ie. M is injective), we can extend our very simple model by introducing some notion of reweighting perceptual dimensions. Rather than inferring category directly from **P** as in eq. 2.5, the features $b_e \in \mathbf{P}$ are reweighted by some weight vector **w** computed as some function W of the representation $\mathbf{p} = M(\mathbf{s})$ and some prior knowledge of the category structure of **C**

$$\mathbf{w} = W(\mathbf{p}, \mathbf{C}) \tag{2.6}$$

$$c_s = max\big(\{p(c_i|\mathbf{p}\cdot\mathbf{w}) : c_i \in \mathbf{C}\}\big) \tag{2.7}$$

Recall that since the features $a \in \mathbf{S}$ are arbitrary, they can include time-varying features, so the weighting function W can, for example, incorporate contextual effects from the recent perceptual past. Category inference being dependent on W has equivalent interpretations in the parlance of artificial neural networks and geometry: as a self-attention mechanism (eg. [79]) giving higher weight to more informative features, or as "collapsing" or "expanding" un/informative dimensions.

And its implications...

The notion of different perceptual features having different weights or importance depending on the acoustic context and the category structure of the phonemes for a particular language is of course far from new.

cue weighting, different types of cues, contextual and informative [72]

A parallel line of thought to the generative models that posit phonetic identity as some positive description of cues or perceptual features are discriminative models that focuses on the features that can be used to tell phonemes apart. A prominent family of discriminative models in phonetics are those that describe a hierarchy of contrastive features[41, 80, 81]. Though they are diverse in their details, in these models M is again typically some fixed feature of the auditory system, and the perceptual space $\mathbf{P}$ that it constructs is some set of high-level descriptions like voicing, frication, or articulator configuration. Typically these features are binary (eg. +/- voiced), rather than continuous.

As an example, consider the proposed contrastive feature hierarchy for russian vowels from [82] (Figure 2.1). Vowel identification is dominated by the primary contrast of +/- back, and successive constrastive features eliminate candidate phonemes until the true phoneme is identified. W's dependence on $\mathbf{C}$ is exemplified (fix passive voice..) by its treatment of "round": -back vowels [e i] are fully determined by +/- high, so for a percept $\mathbf{p}$ with -back, the weight of "round" should be 0. Put another way, the importance of a given feature is dependent on the phonemes that are left ambiguous without it. Any given feature's importance depends on both the set of available features and the set of available categories (the dependence of W, and thus roughly the "meaning" of $\mathbf{P}$, on $\mathbf{C}$ can also be thought of as the "task demand" on phonetic perception, see for example the discussion of [71] in [83]).

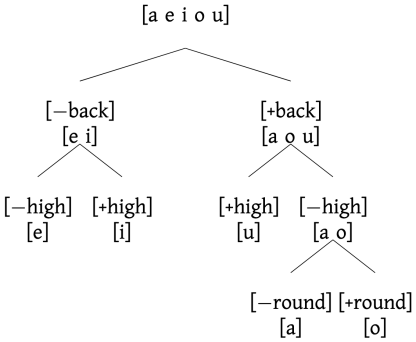


Figure 2.1: Contrastive hierarchy for Russian Vowels, reproduced from [82] without permission

features like +rhotic though don't correspond to anything in the input space tho[84]

expand here on how parameterized stimuli with a single contrast aren't really modeling the problem: like shouldn't it matter how bad the speech sounds sound for claims about natural speech perception? the real question is, during perception, how are the different perceptual axes normalized/selected/weighted; during learning, how does the auditory system learn the space of features? When there is only one feature present the auditory system is performing a qualitatively different task. The use of parameterized stimuli is itself a strong assumption on the nature of the problem that the auditory system is solving. Even parameterizing a family resemblance is so because you assume the weight and salience of different cues. Additionally since there is some "basic cuts" argument to be made about the auditory system and the types of cues that it selects, you're unlikely to hit those if you just use some arbitrary array of stimuli: speech sounds come

pre-optimized for mammalian auditory systems (though obvs mice aren't people) a la adaptive dispersion

"I should emphasize, nevertheless, that there is a great deal of evidence that practice, even large amounts of it, does not produce efficient perception of acoustic alphabets. This is clear, not only in the example of the Morse code, but even more convincingly, perhaps, in the repeatedly unsuccessful attempts to find nonspeech sounds that will work well as part of a reading machine for the bling. Many sound alphabets have been given a thorough trial, but none has proved adequate. It must surely give us pause to know that, while sounds are the universal carriers of language, only one set of sounds — those of speech — serves well."[85] Their conclusions are wrong – that this means that speech is special and has its own processing modality – but the observation does indeed point to the joint optimization of a phonetic space over an auditory space as being constitutive of language, and a potent reason to use speech sounds for category learning.

The notion of the informativeness of different featural dimensions has been given its *fullest* treatment in Keith Kluender and Christian Stilp's application of information theory to phonetic perception[**?** 86, 87, 88]. They summarize their argument, elegantly as always

> "If one's problem is finding the right fencing to corral a unicorn, then there is really no problem at all. Instead the problem is dissolved upon discovery that unicorns do not exist.
>
> Here, we ask the reader to consider the possibility that there are no objects of perception [...]. Like unicorns, they do not exist at all. Instead, there are *objectives* for perception. [...] Perceptual success does not require recovery or representations of the world per se." [**?**]

They argue that the central operation of sensory systems is to adapt to regularity at multiple scales in order to efficiently extract meaningful information from their environment. Rather than a faithful representation of articulatory maneuvers (as in motor theory) or a warped, but still bijective relationship between the acoustic space and perceptual space (as in perceptual warping), they argue that sensory systems discard information that is predictable based on (multiscale) context, and instead represent just the unpredictable, "information-bearing" in an appropriately Shannonistic sense, dimensions.

Though theoretically all configurations of frequencies and amplitudes are possible, naturally produced sounds are strongly constrained by the physics of their production – much of the variation in natural sounds is predictable. Rather than representing the fullness of acoustic variation, the auditory system adapts to redundancies and regularities in sounds to preferentially represent only the unpredictable, informative variation in an "efficient code" [89, 90]. In the case of phonetic perception, where the objective of the listener is to identify the phoneme intended by the speaker rather than perceiving a sound qua sound, the listener attempts to learn auditory features that are maximally informative of phonetic identity[91, 92**?** , 86].

This information-theoretic account provides a mechanism for learning the dimensions of **P** and the form of W. Rather than some a priori, fixed inventory of articulatory/acoustic cues, a listener should learn some set of perceptual features that support the identification of phonemes given the phonemic inventory of their language and the acoustic variability (eg. accent, environment, timbre, etc.) that they are exposed to. Individual listeners do indeed use different combinations of cues with

different weights[93] which are stable over time[94]. Rather than learning some category center and spread over some pre-existing perceptual feature space, the task of the listener is to learn the feature space itself.

The difference between learning **P** and the operation of W is a matter of timescale: over short timescales, W reweights the features in **P** depending on those features that are contextually informative of phonetic identity. While the observation that individual cues are informative, uninformative, and anti-informative depending on the context of surrounding phonemes is a central feature of argument for a motor theory[67], an information-theoretic view interprets this problem as a reweighting of individual features: /s/ differs from /f/ along different featural axes than /s/ differs from /k/, so /s/ shouldn't necessarily rely on the same inventory of acoustic features in all contexts — and particularly when cues are rendered uninformative, the auditory system should adapt to emphasize those that still are(eg. as in [91] and [95]). Contextual effects on phonetic categorization are of course well known (see [**?**]). Where perceptual warping accounts cannot explain results where some or all of the typical acoustic features are replaced, like sine-wave speech[96], noise-vocoded speech[97], or joint spectrotemporal degradation[98]; an information-theoretic view argues that listeners will adapt to use any cues that are still present (as in [91]).

The auditory system does *not* seem to operate in an entirely information-maximizing way when identifying phonemes, however. Consider a category structure like that used by Couchman, Coutinho, and Smith (2010, [99]) depicted in figure 2.2. Each stimulus is composed of four binary features (columns), and stimulus identity is defined by the first feature (0 = category A, 1 = category B). The remaining three features are "epiphenomenal," but stimuli in category B have a greater sum than those in category A. A perfect, information-maximizing observer would learn to only attend to the first dimension, but in speech and many other perceptual categories observers use many, even uninformative dimensions[99, 68] (but see [100]). Non-speech sounds that are strictly uninformative of phonetic identity like pure tones and sweeps can nevertheless strongly influence the perceived phoneme[101, 102], even when the sounds are not immediately adjacent[103]. Such an influence of many, imperfect stimulus dimensions on perception is our signpost to indicate we've arrived back in the bewildering little shire of category structures with family resemblance.

The differing (often implicit) assumptions about the "very complicated model" characterize the major historical disputes in categorical phonetic perception, but also *<suggest the kinds of experiments that might resolve them>.*

expand on each of these: a) Arguably, the careful work of cue theorists led them to motor theories of perception because of a characterization of M as fixed that made the non-invariant acoustic structure of phonetic categories impossible for the auditory system to compute. *but their work was extremely valuable because it explicated the nonlinear nature of acoustic cues and the family resemblance structure of acoustic properties.* b) Work in animal models and infant speech perception demonstrated that phonetic categories were indeed learned (cite infants can acquire all phonemes), but the use of parametric stimuli led to overly-parsimonious models that don't capture the true scope of the problem. need experiments that satisfy the "real problem" (review previous sections and highlight each of the ways the family resemblance structure of phonemes indicates a particular experimental design parameter, but that we need to finish it by adding a neural layer... which we get to in the next section...)

Integrate this into the discussion about infant speech learning research in previous para - The idea that speech acquisition necessarily involves learning the features that are maximally informative is demonstrated by the ability for infants to discriminate between the phonemes of any language, but during language acquisition become specifically attuned to the phonemes of the language(s) they are taught. Though this is typically discussed as learning the statistical regularities of speech sounds (need to cite more because claim of typicality[104][77]), the act of emphasizing the statistical regularity must necessarily mean collapsing those phoentic contrasts that are not present in the language – they aren't informative because no one uses that contrast. indeed they trade off – infants that are better at discriminating the phonemes in their language are worse at discrmiinating those in a non-native language[104] (babies initially can learn all phonemes[77], so they have to learn some feature which necessarily compresses the auditory space[105])

*and focusing on the acquisition of informative stimulus dimensions fundamentally alters the research question. The problem is the mutual translation/misundertanding of what cues *are* – a lot of neurophys research into language ends up using parameterized speech because we want to create parameters and then look for analogies in the brain, either in single neurons or populations. Neuroscientists interpret these cues as 'constitutive' of the phoneme rather than a particular cue describing it (try to find ye old phonetics lit that talks about cue validity as being a problem even in phonetics). This is the pt to turn to 'so instead we need to let the brain reveal its order to us, when presented with a complex array of stimuli, which features does the brain encode and how are they represented???'*

Category A	Category B
0 0 0 0	1 1 1 1
0 1 0 0	1 0 1 1
0 0 1 0	1 1 0 1
0 0 0 1	1 1 1 0

Figure 2.2: Category structure reproduced from [99] without permission. Each stimulus (row of four digits) is composed of four features (columns). Category identity is determined by the first feature (0 = A, 1 = B), but three other "irrelevant" features are present.

Neural mechs

Until now our very simple model has been entirely theoretical, describing the general requirements of the computation of phonetic category identity, but the form of any biological computation is necessarily constrained by the substrate of its implementation (roughly, Marr's levels, for a recent discussion see [106]). Though the model could be retained in its current form by recasting $\mathbf{P}$ as the neural representation of perceptual dimensions from which category $c \in \mathbf{C}$ is inferred, this would require strong assumptions about the form of the neural representation of perceptual dimensions, and in a practical modeling context assumes we have enough information to infer it. To preserve generality at the cost of complexity, we add an additional "layer" to the model,

$$\mathbf{n} = \{n_0, n_i, ...n_{dn} : n \in N^{dm} \subseteq \mathbb{R}^{dn}\} \tag{2.8}$$

where a neural state $\mathbf{n}$, a dn-dimensional instantaneous firing rate of neurons n_i in some neural manifold $\mathbf{N}^{dm}$ of dimension dm embedded within $\mathbb{R}^{dn}$. The manifold embedding N reflects the intrinsic constrants network structure poses on the possible states $\mathbf{n} \in \mathbf{N} \subseteq \mathbb{R}$, but the embedding is arbitrary.

The neural layer is incorporated by modifying equation 2.6 such that

$$M_n = f(\mathbf{s}, \mathbf{p}) : \mathbf{S} \to \mathbf{N} \tag{2.9}$$
$$M_p = f(\mathbf{n}) : \mathbf{N} \to \mathbf{P} \tag{2.10}$$

where some sensory input **s** is mapped to some neural state **n**, which supports some percept **p** from which phonetic category is computed. The dependence of M_n on **p** reflects the possibility of top-down influence on the neural representation of a given stimulus. talk about the representation of time in the model.

note that what we're doing here is largely accounting for incomplete observation and agnosticism of the implementation of perceptual representation. For example there might be some real perceptual dimension that is not independently represented in the neural space, but is computed "downstream" by some structure that we're not observing. In the case of making a claim on the structure of neural representation (talk about alternatives briefly, that not everything necessarily is represented by the firing rate) and full observation, $\mathbf{N} = \mathbf{P}$ *– where* $\mathbf{P}$ *is then the perceptual space represented by the brain from which category identity is computed. So talk about when we separate vs. when we treat them as the same in following section*

more on levels of analysis here? The inextricability of talking about implementation and theory is precisely reflected in the obligation of understanding the ways that the particular system results in the idiosyncracies of the observable behavior – or the degree to which an explanation of the implementation explains and recapitulates the idiosyncracies of the observable behavior is the degree to which it is more or less "correct", in a strict modeling sense. So, precisely for the same reason that we care that our theoretical model accurately describes observable behavior, it is impossible to separate a theoretical model from its implementation – though the temporary illusion is invaluable.

Arguably a computational strategy common to all sensory systems is to exploit regularities in the statistical structure of the natural world to form an efficient sensory representation[107, 108, 89, 88, 109, 110](cite more here bc broad claim). Though the task of phonetic perception is a truly monstrous one (*expand more here?*), work since the heyday of motor theories has demonstrated the remarkable ability of the auditory system to perform the fundamental computations of phonetic categorization has given the problem an air of tractability. And though we still are methodologically limited in our ability to study spech perception in humans at the spatiotemporal scales of its computation, work in animal models as well as recent advances in human brain electrophysiology have given some of the first glimpses.

Several features of our model are happily known to be true of neurons in mammalian auditory cortex.

Neurons in primary auditory cortex jointly encode multiple dimensions of sound[108]. In ferrets presented with an array of stimuli that varied by pitch, timbre, and azimuth[111], more A1 neurons were observed to be sensitive to two or three dimensions (36% and 29%, respectively) than a single dimension (23%). In a subset of neurons, these responses were temporally complex such that the dimensions could be partially recovered by separating sustained from onset responses[112]. Similar results have been observed in marmosets (combined sensitivity to amplitude modulation, frequency modulation, etc. [53]) and in studies that estimated the dimensionality of receptive fields from complex stimuli like dynamic ripples in cats[113]. This is perhaps unsurprising, as cortical neurons being sensitive to multiple dimensions of a stimulus is a trivial reformulation of the well-known hierarchical processing throughout the auditory system (for a review, see [114]): cortical neurons representing "higher order" properties of a stimulus necessarily implies sensitivity to multiple features of the stimulus (provided a generously-enough low-level description of the stimulus feature space).

Maciello and colleagues recently argued that joint, rather than independent encoding of multiple stimulus dimensions is computationally advantageous[115]. Though sensitivity to multiple features makes response patterns ambiguous with respect to the value of any individual dimension, joint encoding provides more information about all represented dimensions to a downstream decoder. If it is the case that joint encoding is constitutive of auditory representations, and individual stimulus or perceptual dimensions are never (or rarely) represented independently, behavior that reflects sensitivity to family resemblance structure rather than optimal rule-based categorization is parsimonious. If all features are estimated simultaneously, influence of "nontarget" dimensions becomes unsurprising.

Auditory cortical neurons adapt to predictable acoustic statistics in order to represent more informative stimulus dimensions at both short and long timescales.

A rich body of research has described the many conditions that auditory representations are modulated by context (for a review, see [116]) at timescales as short as hundreds of milliseconds[117, 118]. Processes like forward masking (cite), stimulus-specific adaptatation (SSA, cite), and suppression of background noise all reflect the general principle that auditory representations adapt to predictable acoustic statistics (cites here) in order to form robust, invariant representations of auditory objects[119] by emphasizing the maximally informative dimensions[113].

Adaptation to noise or stimulus statistics can be characterized as a short-term 'reweighting' of features through processes like synaptic depression[120, 121] or microcircuit interactions[122, 123]. In tasks based on simple parametric sounds, representations of task-relevant stimuli are enhanced on the order of minutes[124]. Animals trained on multiple tasks had neurons that adapted their receptive fields to facilitate the different task demands[125] and reward structures[126]. David and Shamma (2013[127]) argue that short-term integration of auditory context could also be a substrate for representing and comparing auditory features that occur through time.

The auditory system is also plastic on longer timescales to represent the dimensions of sound that are maximally informative to the demands placed on it. Rats trained using a single set of stimuli had differential enhancement of sensitivity to frequency or intensity depending on which they were trained to attend to[128]. Bieszczad and Weinberger observed that such enhancement correlated with the strength of a learned memory trace[129].

speech-specific stuff

The Superior Temporal Gyrus (STG) in humans, or secondary parabelt regions in some other species, of auditory cortex is the primary candidate for representation of higher-order auditory features used in speech perception. Damage to the left posterior Superior Temporal Gyrus, containing BA 22 "Wernicke's area," has long been associated with receptive aphasia, but a variety of human and animal studies have given further insight on the character of speech processing within the STG.

A series of studies from Edward Chang and colleagues recording electrophysiological activity in human temporal lobe using high-density multi-electrode arrays have contributed greatly to our understanding of the encoding of speech sounds, particularly in the superior temporal gyrus (STG) (<- redundancy supreme here)[130].

Recordings of high-gamma (70-150Hz) power show individual electrode sites in middle to posterior STG are selective to acoustically similar groups of phonemes (eg. obstruent vs. sonorant selectivity, plosive vs. fricative selectivity, etc.) in hu-

mans passively listening to natural speech samples[131]. These phonetic sensitivites were reflective of sensitivity to multiple complex acoustic features that are correlated within phonetic categories and that "maximiz[e] vowel discriminability in the neural domain."[131]. Lower frequency (<50Hz) macrocortigraphy recordings also show that subpopulations of pSTG neurons carry information that allows discrimination of consonant-vowel token category analogously to behavioral categorization[132].

In the anterior STG (aSTG), individual sorted units recorded from one person demonstrated complex, speech-specific reponses when one subject was presented with a wide array of sounds[133]. Many (66 of 141) units demonstrated selectivity to one or a few words that was invariant across speaker. Speech selectivity was only partially explained by a linear combination of acoustic features (linear spectrograms and MFCCs), and did not (over-)generalize to noise-vocoded speech, time-reversed speech. Unit responses to individual phonemes also differed by the recent phonetic past, all together suggesting that some units in aSTG are selective to the fine spectrotemporal structure of speech sounds at single-to-few phoneme timescales [133].

Though acoustic response profiles are spatially heterogeneous across the STG and between individuals[131, 134], there does appear to be some functional distinction between anterior and posterior STG with respect to speech sound processing. In macroelectrode recordings in humans listening to natural sentences, pSTG electrodes selectively track phrase-level onsets, while aSTG electrodes have more sustained responses through a phrase. The dissociation between onset and sustained responses was not reflective of the discontinuous vs. continuous nature of consonants and vowels, as selectivity to groups of phonemes (vowels, plosives, nasals, etc.) was mixed in both anterior and posterior STG [134]. Information useful for discrimination of phonetic identity in the pSTG develops and reaches a peak 100-150ms or so after speech sound onset[131, 132], and neural state space projections onto axes representing the activity of neurons sensitive to sound onset or sustained sound show a reliable sweep between posterior and anterior STG on the order of seconds. summarize description of temporal processing distributed across multiple regions that potentially reflects different parts of the information being reflected in different... codes.

Animal research of neural mechanisms of speech sound processing is quite sparse, and so our understanding is relatively coarse and by analogy from more general auditory research. Speech training in rats evokes a complex set of changes to acoustic response properties in several auditory cortical fields loosely analogous to secondary cortical areas in humans[135]. Neurons in the anterior auditory field (AAF) and A1 were more responsive to the initial consonant in consonant-vowel (/CV/) pairs in trained vs. control rats (27% and 57% more spiking activity, respectively). Additionally, the proportion of neurons that were responsive to 2kHz tones (the spectral peak in the speech tokens used) increased by 65% in AAF and 38% in A1 after speech training compared to control rats. In contrast, in response to vowels VAF and PAF were less responsive following speech training (42% and 30% fewer spikes, respectively, vs. controls). In neurons that had similar frequency tuning, responses to consonants were more correlated in AAF and VAF, and responses to vowels were less correlated in AAF, A1, and VAF after speech training (vs. controls)[135].

These results[135] may not establish definitive roles for secondary auditory fields in rodent auditory cortex, but in sum do suggest that speech training induces long-lasting plasticity in auditory cortex, and suggests that processing may be distinct for different acoustic features in anterior vs. posterior fields as in humans. Mice trained

to discriminate speech sounds were returned to chance following lesions of auditory cortex[136], indicating its necessity. Task-specific plasticity[137] and contribution to processing task-relevant auditory stimulus categories[138] has been previously demonstrated in AAF, which is thought to operate as a parallel processing system, with response latencies comparable to or lower than A1 in cats[139] and mice[140]. PAF is a secondary auditory cortical area and thought to be downstream from both A1 and AAF[141, 142]. Though their functional specialization of computational role might not be equivalent in humans, it is parsimonious to assume that primary and secondary auditory cortical areas in nonhuman mammalian auditory systems process acoustic information in such a way that supports the recognition of phonetic identity.

Talk about the categorical decisionmaking process downstream, implications for role of nonauditory, frontal, etc. zones that actually do the integration with syntactical, semantic information. Differentiate that we're concerned about the derivation of the acoustic level, the perceptual dimensions that facilitate, but may not constitute the identity of a phoneme.

uh is there a name for the conclusion of an introduction because i need to make a section break to write it lmao

In lightly constraining the constitution of **N**, loosely the neural "representation" of phonetic information, the human and animal results hint at the dissociation between **N** and **P** in our model — en passant to *the statement of the research problem.*

Suppose that one dimension $b_{vot} \in \mathbf{P}$ is the voice onset time, which dissociates voiced from unvoiced consonants (eg. /b/ vs. /p/) as the time between the onset of phonation and the occlusion of the stop. Further suppose a neural system analogous to the temporal landmark model suggested by [134] where the high-energy plosive of the occlusion is "encoded" by the activity of some region analogous to the phrase-onset sensitivity of pSTG, and the sonorant, spectral quality of the voicing is encoded by another region. In this scheme, some downstream region (really need to give a name to the "readout" part of the model) infers VOT by comparing the relative timing of gross spiking activity between these two regions. In this hopelessly naïve instantiation of our model, the dimension b_{vot} is some real-valued (though not necessarily linear) value from negative to positive voice onset times. Such a dimension is not present in **N** as characterized by the n-dimensional space of, say, instantaneous firing rate of n neurons, requiring M_p, the mapping between them.

The dissociation of the descriptions of **N** and **P** thus, in our model, defines the research problem:

1) We characterize the problem the brain faces in auditory phonetic perception is to learn some perceptual space **P** of maximally informative perceptual dimensions that supports the identification of received phonemes by flexibly adapting to the information present in the phoneme.

2) Understanding the neural mechanisms of auditory phonetic perception is describing the way **P** is implemented by some neural state manifold **N** in such a way that the difference between **N** and **P** is minimized.

It is not necessarily the case that we should expect to find neurons, or even collections of neurons, whose time-averaged firing rate is the literal measurement of the

perceptual dimensions used to compute phonetic identity. We also don't expect to be able to estimate the full manifold of all neurons that are involved with the process, so there will ultimately always be some gap between **P** and **N**. Roughly, kept independent, **P** is the level of "representation" — the basis from which the brain derives its use of phonetic information (though we dont' characterize **P** as the unique source of information, as information is represented at multiple scales (syntactic, semantic) and is bidirectional (predictive as well as receptive)) — while **N** is the level of "implementation". Finding a "neural representation" of phonemes is thus describing the implementation of **P** by **N**, ergo constructing them in a such a way that their difference is minimized.

This distinction may read as trivial, but it precludes a majority of the common methodological kinks of contemporary cognitive neuroscience. The implicit assumption of "decoding"-based analysis strategies is that neural representation is encoded in the language of time-averaged firing rate, and that the accuracy of some (usually uninterrogated) classification algorithm on the timeseries of firing rates (or BOLD level, or EEG bandpass amplitude, etc.) is reflective of the presence or absence of category information in the data. The same assumption is made in the case of so-called "Representational Similarity Analysis," and any number of other analytical ruts that uncritically characterize the geometry of the brain and the perceptual reality it supports as euclidean spaces with the axes of whatever recording methodology is handy for the dataset.

In both, the geometry of the perceptual space is also typically uninterrogated, where the parameters that were used to synthesize the stimuli, or the category labels imposed by the researcher are analyzed as if they were faithfully represented by the brain. This, despite the creation of non-isomorphic representations of physical phenomena being the entire goal of efficient perception (see [87, 143]) — if representation operated like an isomorphism then perceptual learning would be entirely unnecessary.

Rather than assuming the perceptual structure of phonemes by prespecifying cues and synthesizing sounds, or assuming the representational language of the brain to be time-averaged firing rate, we take the role of empirical geometers and attempt to preserve as much of the natural complexity of the problem and derive both from the data.

2.3 Specific Aims

Scraps

- Segmenting strategies [144]
- Scrambled vs. unscrambled sounds? (cites 12, 18, and 25 in [145])
- inferring perception-action loops from data [146]
- complementary roles of cell types and manifold dynamics [147]
- LFADS for sequential autoencoders [148]
- modeling auditory waveform with kernels [89]
- brain is actually a dynamic system and need to model the manifold [149] becasue the same brain region does multiple things at the same time with the manifold lol [150]
- ?time constant of auditory sensitivity in STG neurons?

- The natural analog of the philosophical problem of universals in the conditioning paradigm is stimulus generalization [151]
- Neural nets for estimating nonlinear STRFs, se [108]
- extracting maximally informative features [92]
- creating superstimuli [152]
- estimating nonlinear STRF[153]
- remember to return to shepard and 2nd order isomorphism stuff

  The history of this question includes Shepard and Tversky's multidimensional scaling and its criticisms, and also extends through Shepherds' "second-order isomorphisms" (cite representation is representation of similarity)

arguably the cue-theorists arrived at the wrong conclusions was because of their belief about the innateness of the auditory-perceptual mapping: it must have been genetic, so therefore language is parsimoniously some special module, etc. etc. Research based on synthesized parameters based on cues then carry that error further by not representing the full scope of the problem. like how they eventually discarded the notion of cues (definitely need more detail in that story about specific examples of how cues are conflicting in different contexts) was because they considered their interaction with other cue dimensions. If we instead take the info-theoretic perspective seriously then learning a phoneme should be the act of learning the maximally informative dimensions. since we see individual differences in cue weighting within individuals, we would also expect people's dimensions to be different... but if there is only one or a few carefully parameterized dimensions of variation present in the stimulus set, of course they'll learn those, so we need to instead use a stimulus set that preserves as much of the natural variation within category as possible and allow the animals to learn the contrastive dimensions themselves. using only two categories is of course a simplification, but it still mimics at least the nature of the learning problem in qualitative form, and also [evidence that infants learn stop consonant boundaries early and they are primary and near-universal across languages indicating that they are sorta self-stable system where the big featural distinction of being stops makes it so they are like a 'submodule' within a phonetic set.]

parameterized vs natural speech is actually reflective of a much larger positivist/-naturalist philosophical divide – they presuppose by testing a parameter of category membership, but postiive evidence is not evidence that parameter is actually constitutive of the category itself – for example if you had two categories "games" and "cars," "weight" might be a reasonably good way to assign category membership, but it is not at all the only, or even the most salient difference between those categories. Like i feel like I'm crazy sometimes because shouldn't the fact that synthesized speech sounds sound bad be a problem? They might have all the theoretical justification in the world but the fact that they so badly imitate what even a plausible phoneme would sound like should be like a red flag for the generalizability of the conclusions that can be drawn from them.

theoretical problems with simplified stimuli - low-dimensional and linearly-separable stimulus spaces are fundamentally different than the high complexity of naturalistic stimuli... for all we know the computations are just straight up not comparable! [154]

behavior

If the objective of the listener is to understand, ie. to be able to parse the speech sounds made by their interlocuter, then how is that different than that of the mouse, which is to get water? They are

identical when water is only given when knowledge is demonstrated, but that is impossible when the chance of false positive is 50%. more importantly how that intersects with passive learning/non-rewarded phoeme studies.

reasons for speech stimuli: category complexity depends on the density of the space. the competition for desire for rich vocabulary of phonemes with limited articulatory palette means that we need to fit a shitload of acoustic complexity into an extremely small temporal window with a small amount of potential variation. So yeah parameterized mouse calls might work but that's like a feature of the density of the communication space, but they also have extremely subtle cues in their environment that they need to parse... so speech sounds are good because they're not species-specific but also because they're stimuli that we know have a potential subjective categorization structure but one that is sufficiently complex. speech sounds also take advantage of the innate contours of the auditory system,

trying a fresh rewrite: q: why use natural speech rather than some other synthesized, complex, high-dimensional acoustic stimulus? a: though the question is about auditory category learning in general, the auditory system is not some lockean tabula rasa because natural law dictates that auditory reality isn't some equiprobable playground where all sounds are possible. the auditory system evolved to be better able to learn certain acoustic contrasts compared to others because the fact that some contrasts are more informative than others is written into the very sinew of natural law (cite patricia kuhl's 'basic cuts' argument, tony zador 'critique of pure learning'). it is also not sufficient to identify one or a few of these 'natural auditory-perceptual gradients' and synthesize stimuli along them: the problem that languages have been solving for <many> years is how to pack many contrasts that are all mutually intelligible at rapid timescales (low ... resolution?) across those gradients. Close phonetic contrasts are thus complex stimuli optimized to be discriminable by the mammalian auditory system in a dense category-space, making the reliance on the family resemblance-type structure (rather than a simple rule-based solution) that typifies phonetic identification and other complex category processing necessary

The requirement for doing it online is because what you're doing is doing a much more efficient exploration of the massive massive stimulus space– theoretically if you freaking play a billion phonemes of infinite variation you will just be grid searching all the same space that you would by presenting it online. Sooooo if we can't make online stimulus modulation work, then we just need to make sure we have sufficient samples to tile the space. Importantly though, since we're not necessarily trying to explain speech as such, but rather then learning of some general auditory categories, the degree to which our stimuli (and thus our estimates of perceptual dimension) only really affects the degree to which we simulate the problem of speech. What could be degraded? well, it could be the case that we use too few stimuli to have a sufficiently complex categorization in the first place, but that's pretty unlikely because of the extreme variability of speech across vowel contexts, let alone speakers. Fitting after training, or like even online fitting, or even just like testing their responses to generated stimuli afterwards is totally valid as a test of the validity of the dimensions.

imaging

analysis & modeling

Neuroscientists sorta blithely assume what the features of a stimulus are, from the seemingly harmless and physically based – frequency, direction, angle, etc. – to the absurd – rsa et al. But these dimensions rarely behave like 'real' perceptual dimensions [155] – the transforma-

tion is actually the critical part.

assuming feature dimensions is always a bad assumption – eg what features have the metric structure that measure similarity/dissimilarity of rectangles? [155]

Lots of people already talking about this, but even criticisms sorta treat perceptual dimensions as a given, and it is the brain's fault that it doesn't represent them. [**?**]

2.4 Significance & Broader Impacts

2.5 Notes

Bailey & Summerfield - 1980

A perceptual system in which the information for phonetic perception was a set of cues would have to incorporate three kinds of knowledge if it were to function successfully. It would have to know, first, which aspects of the acoustic signal are cues and which are not; second, it would need to possess a sensitivity to the pattern of cooccurrence of cues for each phone in its perceptual repertoire; third, it would need to appreciate the proper temporal coordination of the cues within each pattern. There is no reason, in principle, why a device could not be built to perceive phonetic identity from a substrate of acoustic cues, provided it was endowed with an articulatory representation sufficient to embody these three kinds of knowledge. However, we doubt that such a system could evolve in the natural world. For a species to acquire a knowledge of articulatory constraints, it would be necessary first that information specifying those constraints be available for the species, and second that the species possess a prior sensitivity to that information. The knowledge that a particular set of cues combine to indicate the presence of a given phone could be acquired in either of two ways. The identity of the phone could be specified independently of the set of acoustic cues, but this would hardly solve the problem and would preempt the need to evolve a sensitivity to the cues. Alternatively, the signal could specify directly both the identity of the cues and their temporal coordination, but then information in the signal that specified the coherence of its elements would, isomorphically, specify the articulatory event from which that coherence derived. However, the presence of this information about articulation in the signal, and a predisposition to register it on the part of the perceiver, would obviate the need for any internalized articulatory referent to mediate the acoustic-phonetic translation.

These considerations lead us to question the validity of equating the operational and functional definitions of an acoustic cue. A cue was defined operationally as a physical parameter of a speech signal whose manipulation systematically changes the phonetic interpretation of the signal. Although it is clear that perceptual sensitivity must exist to the consequences of manipulating a cue, it is not necessary to suppose that the cue is registered in perception as a discrete functional element.[67]

2.6 meta

to-read

- revisit the tversky lit and check Danielle's cites for more
- the long-term imaging/ephys papes
- [115]
- [156]
- [157]
- [146]
- [158]
- [150]
- [159]
- [160]
- [161]
- [162]
- [163]
- [164]
- [165]
- [166]
- [167]
- [168] - methods
- [169] - methods
- [144] - methods

bookmarks

- [147] - p6

scraps - neuro

- auditory processing as domain-general and domain-specific across multiple timescales [145]
- abrupt transitions, at least in neural data [156]
- multimodal representations and preserved neural manifold dynamics across inference tasks in M1 [150]

- timescales of processing expand across auditory hierarchy (and more generally have different timescales of integration and lags) [145] and are lateralized [170]
- contributions from basal ganglia in reward learning for acoustic dimentions [171]
- this bif ol review [172]
- neurons that process auditory information at phonetic timescales are relatively insensitive to spectral quality [145]
- find where this go -> Indeed different people have different cue weightings that are more or less adaptive[173]
- emergence of invariant reps in secondary auditory cortex[174]
- vocalization sensitive neurons in anterior left acx with different projection patterns from/to L6 that are experience dependent. (cfos[170])

scraps -

Part II

Autopilot

Neuroscience needs behavior, and behavioral experiments require the coordination of large numbers of heterogeneous hardware components and data streams. Currently available tools strongly limit the complexity and reproducibility of experiments. Here we introduce Autopilot, a complete, open-source Python framework for experimental automation that distributes experiments over networked swarms of Raspberry Pis. Autopilot enables qualitatively greater experimental flexibility by allowing arbitrary numbers of hardware components to be combined in arbitrary experimental designs. Research is made reproducible by documenting all data and task design parameters in a human-readable and publishable format at the time of collection. Autopilot provides a high-level set of programming tools while maintaining submillisecond performance at a fraction of the cost of traditional tools. Taking seriously the social nature of code, we scaffold shared knowledge and practice with a publicly editable semantic wiki and a permissive plugin system. Autopilot's flexible, scalable architecture allows neuroscientists to work together to design the next generation of experiments to investigate the behaving brain.


DOCS


SOURCE


WIKI


PAPER SOURCE


PAPER PLUGIN

We would like to acknowledge and thank Lucas Ott and Tillie Morris for doing most of the behavioral training and being so patient with the bugs, Brynna Paros and Nick Sattler for their help with constructing our behavioral boxes, Chris Rogers who has been brave enough to adopt and contribute to Autopilot in its roughest state, Arne Meyer, Mikkel Roald-Arbøl, and David Robbe who have contributed code and advice, Mackenzie Mathis, Alex Mathis, Gonçalo Lopes, and Gary Kane who collaborated on the DeepLabCut-Live project and provided many a mentorship along the way, Jeremy Delahanty for his inspiring tenacity and humbleness in thinking about better research tools, Matt Smear and Reese Findley for loaning us their Bpod for far longer than they intended to, John Boosinger and the rest of the staff in the machine shop for all their advice and letting me use all their tools, Erik Flister whose Ratrix software inspired some of the design features of Autopilot [175], Santiago Jaramillo whose TASKontrol[176] gave inspiration for GUI design and served as an early scaffolding to learn Python, my labmates Molly Shallow and Sam Mehan who kept me afloat in my last months of dissertation writing, Rocky Penick for her help strapping Autopilot onto the back of Evan Vickers' mesoscope rig (and Evan for letting us play with his rig), several artists on `flaticon.com` *(Freepik, Nikita Golubev, Those Icons) whose work served as stems for some of the figures, and the Janet Smith House for the endless support and relentless criticism of the figures. This material is based on work supported by NIH NIDCD R01 DC-015828, NSF Graduate Research Fellowship No. 1309047, and a University of Oregon Incubating Interdisciplinary Initiatives award.*

Contribution Statement: JLS designed and wrote the software, documentation, wiki, figures, ran the tests, and wrote and edited the paper. LO trained the animals and did an extensive amount of beta testing, bugfinding, and made some of the hardware designs on the wiki. MW mentored, edited the paper, and beta tested the software.

3

Introduction

Animal behavior experiments need precision and patience, so we make computers do them for us. The complexity of contemporary behavioral experiments, however, presents a stiff methodological challenge. For example, researchers might wish to measure pupil dilation[177], respiration[178], and running speed[179], while tracking the positions of body parts in 3 dimensions[180] and recording the activity of large ensembles of neurons[181], as subjects perform tasks with custom input devices such as a steering wheel[182] while immersed in virtual reality environments using stimuli synthesized in real time[183, 184]. Coordinating the array of necessary hardware into a coherent experimental design—with the millisecond precision required to study the brain—can be daunting.

Historically, researchers have developed software to automate behavior experiments as-needed within their lab or relied on purchasing proprietary software (eg. [185]). Open-source alternatives have emerged recently, often developed in tandem with hardware peripherals available for purchase [186, 187]. However, the diverse hardware and software requirements for behavioral experiments often lead researchers to cobble together multiple tools to perform even moderately complex experiments. Understandably, most software packages do not attempt to simultaneously support custom hardware operation, behavioral task logic, stimulus generation, and data acquisition. The difficulty of designing and maintaining lab-idiosyncratic systems thus defines much of the everyday practice of science. Idiosyncratic systems can hinder reproducibility, especially if the level of detail reported in a methods section is sparse[188]. Additionally, development time and proprietary software are expensive, as are the custom hardware peripherals that are required to use most available open-source behavior software, stratifying access to state-of-the-art techniques according to inequitable funding distributions.

Technical challenges are never merely technical: they reflect and are structured by underlying *social* challenges in the organization of scientific labor and knowledge work. Lab infrastructure occupies a space between technology intended for individual users and for large organizations: that of *groupware*[1] [190, 189]. Experimental frameworks thus face the joint challenge of technical competency while also embedding in and supporting existing cultures of practice. Behind every line of code is an unwritten wealth of technical knowledge needed to make use of it, as well as an unspoken set of beliefs about how it is to be used — labs aren't born fresh on release day ready to retool at a moment's notice, they're held together by decades of duct tape and run on ritual. The boundaries of this "contextual knowledge" extend fluidly beyond individual labs, structuring disciplinary, status, and role systems in scientific work[191]. Given their position at the intersection of scientific theory, technical work, data production, and social organization, experimental frameworks are an elusive design challenge, but also an underexplored means of realizing some of our loftier dreams of open, accessible, and collaborative science.

[1] "Our original definition of groupware was 'intentional group processes plus software to support them.' It has both *computer* and *human* components: software of the computer and 'software' of the people using it. [...] Recently this definition has been extended to include other more expressly cultural factors including myth, values and norms. The computer software should reflect and support a group's purpose, process and culture."

Peter and Trudy Johnson-Lenz (1991)[189]

Here we present Autopilot, a complete open-source software and hardware framework for behavioral experiments. We leverage the power of distributed computing using the surprisingly capable Raspberry Pi 4[2] to allow researchers to coordinate arbitrary numbers of heterogeneous hardware components in arbitrary experimental designs.

[2] See Table 5.2

Autopilot takes a different approach than existing systems to overcome the technical challenges of behavioral research: *just use more computers*. Specifically, the advent of inexpensive single-board computers (ie. the Raspberry Pi) that are powerful enough to run a full Linux operating system allows a unified platform to run on every Pi or other computer in the system so that they can work together seamlessly. At the core of its architecture are networking classes (Section 5.8) that are fast enough to stream electrophysiological or imaging data and flexible enough to make the mutual coordination of hardware straightforward.

This distributed design also makes Autopilot extremely scalable, as the Raspberry Pi's $35-$75 price tag makes it an order of magnitude less costly than comparable systems (Section 4.3). Its low cost doesn't come at the expense of performance or useability: Autopilot provides an approachable, high-level set of tools that still have input and output precision between dozens of microseconds to a few milliseconds (Sections 4.1 and 6).

Autopilot balances experimental flexibility with support. Its task design infrastructure is flexible enough to perform arbitrary experiments, but also provides support for data management, plotting task progress, and custom training regimens. We try to bridge multiple modalities of use: use its modular framework of tools out of the box, or use its complete low-level API documentation[3] to hack it to do what you need. Rather than relying on costly proprietary hardware modules, users can take advantage of the wide array of peripherals and extensive community support available for the Raspberry Pi. Autopilot is designed to be *permissive*: build your whole experiment with it or just use its networking modules, adapt it to existing hardware, integrate your favorite analysis tool. We designed Autopilot to *play nice* with other software libraries and existing practices rather than force you to retool your lab around it.

[3] https://docs.auto-pi-lot.com

Finally, we have designed Autopilot to help scientists do reproducible research and be good stewards of the human knowledge project. Experiments are not written as scripts that are reliant on the particularities of each researcher's hardware configuration. Instead, we have designed the system to encourage users to write reusable, portable experiments that can be incorporated into a public central library while also allowing space to iterate and refine without needing to learn complicated programming best-practices to contribute. Every parameter that defines an experiment is automatically saved in publication-ready format, removing ambiguity in reported methods and facilitating exact replication with a single file. Its plugin system is built atop a densely-linked semantic wiki[4] that fluidly combines human- and computer-readable, communally editable technical knowledge that surrounds your experiments with the software that performs them.

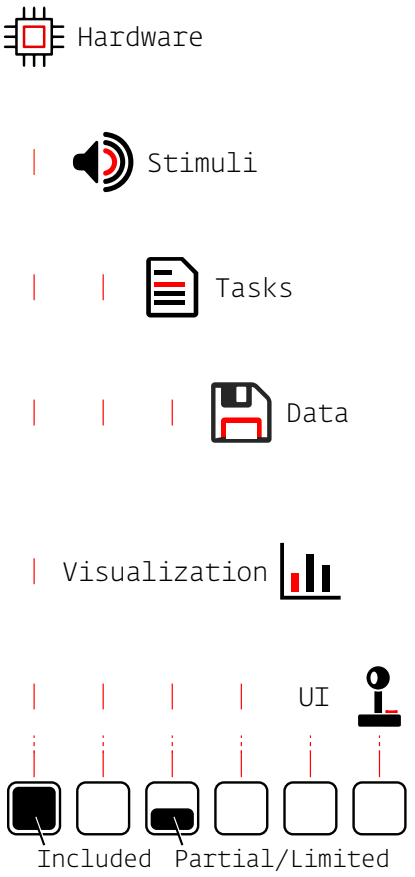
[4] https://wiki.auto-pi-lot.com

We begin by defining the requirements of a complete behavioral system and evaluating two current examples (Sections 3.1 and 3.2). We then describe Autopilot's design principles (Section 4) and how they are implemented in the program's structure (Section 5). We close with a demonstration of its current capabilities and our plans to expand them (Sections 6 and 7).

3.1 Existing Systems for Behavioral Experiments

At minimum, a complete system to automate behavioral experiments has 6 requirements:

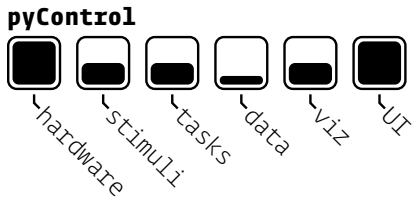
1. Hardware to interact with the experimental subject, including **sensors** (eg. photodiodes, cameras, rotary encoders) to receive input and **actuators** (eg. lights, motors, solenoids) to provide feedback.
2. Some capability to synthesize and present sensory stimuli. Ideally both discrete stimuli, like individual tone pips or grating patches, and continuous stimuli, like those used in virtual reality experiments, should be possible.
3. A framework to coordinate hardware and stimuli as a task. Task definition should be flexible such that it facilitates rather than constrains experimental design.
4. A data management system that allows fine control of data collection and format. Data should be human readable and include complete metadata that allows independent analysis and reproduction. Ideally the program would also allow some means of realtime data processing of sensor values for use in a task.
5. Some means of visualizing data as it is collected in order to observe task status. It should be possible to customize visualization to the needs and structure of the task.
6. Finally, a user interface to control task operation. The UI should make it possible for someone who does not program to operate the system.



We will briefly describe two other systems that meet this definition of completeness: pyControl and Bpod.

pyControl

pyControl[192] is a behavioral framework built in Python by the Champalimaud Foundation. It uses the micropython microcontroller ("pyboard") as its primary hardware device along with several extension boards sold by openephys. The pyboard has four I/O ports, or eight with a multiplexing expander board. Schematics are available for many other hardware components like solenoid valve drivers and rotary encoders. Multiple pyboards can be connected to a computer via USB and run independent tasks simultaneously with a GUI.



There is limited support for some parametrically defined sound stimuli, presented from a separate amplifier connected using the I2C protocol. Visual stimuli are unsupported.

Like most behavioral software, pyControl uses a finite-state machine formalism to define its tasks. A task is a set of discrete states, each of which has a set of events that transition the task from one state to another. pyControl also allows timed transitions between states, and one function that is called on every event for a rough sort of parallelism. pyControl also allows the use of external variables to control state logic, making these state machines more flexible than strict finite state machines.

```
D 0 2
D 8976 3
D 8976 1
P 8976 Print Statement
D 10162 3
D 10163 2
```

Figure 3.1: pyControl data is stored as plain text, each line having a type (like **D**ata or **P**rint), timestamp, and state

All events and states are stored alongside timestamps as a plain text log file, one file per subject per session (Figure 3.1). Analog data are stored in a custom binary serialization that alternates 4-byte data and timestamp integers.

There is only one plot type available in the GUI, a raster plot of events, and no facility for varying the plot by task type. The GUI is otherwise quite capable, including the ability to batch run subjects, redefine task variables, and configure hardware.

Bpod

Bpod is primarily a collection of hardware designs and an assembly service run by Sanworks LLC. Similar to pyControl, each Bpod behavior box is based on a finite-state machine microcontroller with four I/O ports. Additional hardware modules provide extended functionality. Bpod is controlled using its own MATLAB package, though there are at least two other third-party software packages, BControl and pyBpod, that can control Bpod hardware. A task is implemented as a MATLAB script that constructs a new state machine for each trial, uploads it to the Bpod, and waits for the trial to finish. As a result, only one Bpod can be used per host computer, or at least per MATLAB session. Data are stored as trial-split events in a MATLAB structure.

There are a few basic plots for two-alternative forced choice tasks, but there doesn't seem to be a prescribed way to add additional plots. Bpod has a reasonably complete GUI for managing the hardware and running tasks, but it is relatively technical (Figure 3.2).

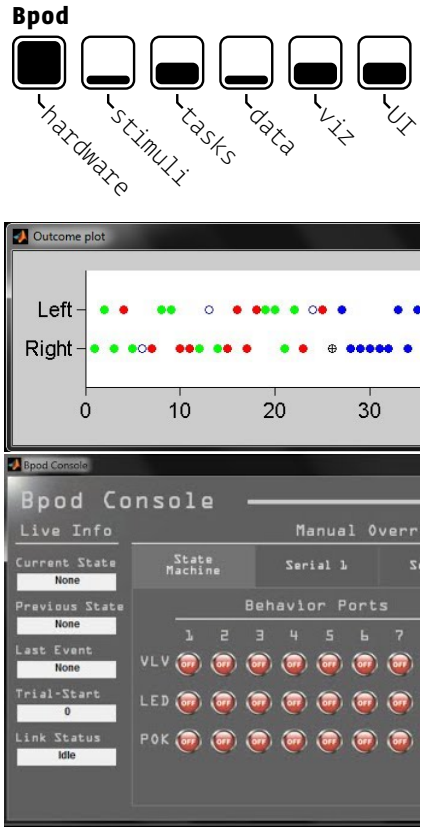


Figure 3.2: A Bpod event plot (above) showing the results of individual behavioral trials, and the Bpod GUI (below).

For brevity we have omitted many other excellent tools that perform some subset of the operations of a complete behavioral system, or otherwise have a substantial difference in scope.[5]

3.2 *Limitations of Existing Systems*

We see several limitations with these and other behavioral systems:

- **Hardware** - Both Pycontrol and Bpod strongly encourage users to purchase a limited set of hardware modules and add-ons from their particular hardware ecosystem. If a required part is not available for purchase, neither system provides a clear means of interacting with custom hardware aside from typical digital inputs and outputs, requiring the user to 'tack on' loosely-integrated components. There is also a hard limit on the *number* of hardware peripherals that can be used in any given task, as there is no ability to use additional pyboards or Bpod state machines in a single task. The microcontrollers used in these systems also impose strong limits on their software: neither run a full, high-level programming language[6]. We will discuss this further in section 4.2. A broader limitation of existing systems is the difficulty of flexibly integrating diverse hardware with the analytical tools necessary to perform the next generation of behavioral neuroscience experiments that study "naturalistic, unrestrained, and minimally shaped behavior"[200].

[5] **Other tools:**
- Bonsai[193] - site, git
- Expyriment[194] - site, git
- PsychoPy[195] - site, git
- OpenSesame[196] - site, git
- SMiLE - docs
- ArControl[197] - git
- and see OpenBehavior

[6] Bpod runs custom firmware written in C++ on a Teensy 3.6 microcontroller. pyControl's pyboard runs micropython, a subset of Python that excludes canonical libraries like numpy[198] or scipy[199]

- **Stimuli** - Stimuli are not tightly integrated into either of these systems, requiring the user to write custom routines for their synthesis, presentation, and description in the resulting data. Neither are capable of delivering visual stimuli. Since the publication of the initial version of this manuscript, Bpod has added support for a HiFiberry sound card that we also describe here[201], but the sound generation API appears to be unchanged, with a single method for generating sine waves. Some parametric audio stimuli are included in the pyControl source code but we were unable to find any documentation or examples of their use.

- **Tasks** - Tasks in both systems require a large amount of code and effort duplication. Neither system has a notion of reusable tasks or task 'templates,' so every user typically needs to rewrite every task from scratch. Bpod's structure in particular tends to encourage users to write long task scripts that contain the entire logic of the task including updating plots and recreating state machines (Figure 3.3). Since there is little notion of how to share and reuse common operations, most users end up creating their own secondary libraries and writing them from scratch. Another factor that contributes to the difficulty of task design in these systems is the need to work around the limitations of finite state machines, which we discuss further in section 5.3.

- **Data** - Data storage and formatting is basic, requiring extensive additional processing to make it human readable. For example, to determine whether a subject got a trial correct in an example Bpod experiment, one would use the following code:

  ```
  SessionData.RawEvents.Trial{1,1}.States.Punish(1) ~= NaN
  ```

  As a result, data format is idiosyncratic to each user, making data sharing dependent on manual annotation and metadata curation from investigators.

- **Visualization & GUI** - The GUIs of each of these systems are highly technical, and are not designed to be easily used by non-programmers, though pyControl's documentation offsets much of this difficulty. Visualization of task progress is quite rigid in both systems, either a timeseries of task states or plots specific to two-alternative forced choice tasks. In the examples we have seen, adapting plots to specific tasks is mostly ad-hoc use of external tools.

- **Documentation** - Writing good documentation is challenging, but particularly for infrastructural systems where a user is likely to need to modify it to suit their needs it is important that it be possible to understand its lower-level workings. PyControl has relatively good user documentation for how to use the system, but no API-level documentation. Bpod's documentation is a bit more scattered, and though it does have documentation for a subset of its functions, there is little indication of how they work together or how someone might be able to modify them.

- **Reproducibility** - As of November 2020, pyControl has versioned task files that append a hash to each version of a task and save it along with any produced data, tying the data to exactly the code that produced it. PyControl's most recent releases have explicit version numbers, but these don't appear to be saved along with the data. Bpod stores neither code nor task versions in its data. Neither system saves experimental parameter changes by default —and the GUIs of both allow parameters to be changed at will— and so critical data could be lost and experiments made unreproducible unless the user writes custom code to save them.

```
for currentTrial = 1:MaxTrials
% new state machine every trial
sma = NewStateMachine();

% add states and transitions
sma = AddState(sma,
    'Name', 'Wait', ...
    'Timer', 0, ...
    'StateChangeConditions', ...
    {'Port2In', 'Delay'}, ...
    'OutputActions', ...
    {'AudioPlayer1','*'});
% add more states ...

% upload and run task
SendStateMatrix(sma);
RawEvents = RunStateMatrix;

% manually gather data and params
BpodSystem.Data = AddTrialEvents(
    BpodSystem.Data, RawEvents);

% plotting in the main loop
UpdateSideOutcomePlot( ... );
UpdateTotalRewardDisplay( ... );

% manually save data
SaveBpodSessionData;
end
```

Figure 3.3: Bpod's general task structure.

Bpod has an undocumented plugin system, but neither system has a formal system for sharing plugins or task code, requiring work to be duplicated across all users of the system.

- **Integration and Extension** - Integration with other systems that might handle some out-of-scope function is tricky in both of these example systems. All systems have some limitation, so care must be taken to provide points by which other systems might interact with them. One particularly potent example is the use of Bpod in the International Brain Laboratory's standardized experimental rig[202], which relies on a single-purpose 93 page PDF to describe how to use the iblrig library, which consists of a large amount of single-purpose code for stitching together pybpod with bonsai for controlling video acquisition. Even if a system takes a large amount of additional work to integrate with another, hopefully the system allows it to be done in a way such that it can be reused and shared with others in the future so they can be spared the trouble. The relatively sparse documentation and the high proportion of ibl-specific code present in the repository make that seem unlikely.

Some of these limitations are cosmetic—fixable with additional code or hardware—but several of the most crucial are intrinsic to the design of these systems.

These systems, among others, have pioneered the development of modern behavioral hardware and software, and are to be commended for being open-source and highly functional. One need look no further for evidence of their usefulness than to their adoption by many labs worldwide. At the time that these systems were developed, a general-purpose single-board computer with performance like the Raspberry Pi 4 was not widely available. The above two systems are not unique in their limitations[7], but are reflective of broader constraints of developing experimental tools: solving these problems is *hard*. We are only able to articulate the design principles that differentiate Autopilot by building on their work.

[7] And Autopilot, of course, also has many of its own weaknesses

4

Design

Autopilot distributes experiments across a network of Raspberry Pis,[1] a type of inexpensive single-board computer.

[1] Raspberry Pi model 4B, see Table 5.2

Autopilot has three primary design principles:

1. **Efficiency** - Autopilot should minimize computational overhead and maximize use of hardware resources.
2. **Flexibility** - Autopilot should be transparent in all its operations so that users can expand it to fit their existing or desired use-cases. Autopilot should provide clear points of modification and expansion to reduce local duplication of labor to compensate for its limitations.
3. **Reproducibility** - Autopilot should maximize system transparency and minimize the potential for the black-box of local reprogramming. Autopilot should maximize the information it stores about its operation as part of normal data collection.

4.1 Efficiency

Though it is a single board, the Raspberry Pi operates more like a computer than a microcontroller. It most commonly runs a custom Linux distribution, Raspbian, allowing Autopilot to use Python across the whole system. Using an interpreted language like Python running on Linux has inherent performance drawbacks compared to compiled languages running on embedded microprocessors. In practice these drawbacks are less profound than they appear on paper: Python's overhead is negligible on modern processors[2], jitter and performance can be improved by wrapping compiled code, etc. While we view the gain in accessibility and extensibility of a widely used high-level language like Python as outweighing potential performance gains from using a compiled language, Autopilot is nevertheless designed to maximize computational efficiency.

[2] and improvements to CPython in Python 3.11 and onwards will bring overhead close to zero[203]

Concurrency

Most behavioral software is single-threaded (Figure 4.1), meaning the program will only perform a single operation at a time. If the program is busy or waiting for an input, other operations are blocked until it is finished.

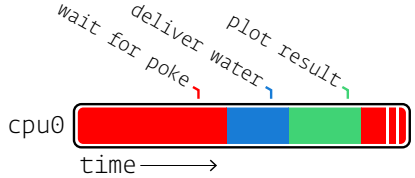


Figure 4.1: A single-threaded program executes all operations sequentially, using a single process and cpu core.

Autopilot distributes computation across multiple processes and threads to take advantage of the Raspberry Pi's four CPU cores. Most operations in Autopilot are executed in **threads**. Specifically, Autopilot spawns separate threads to process messages and events, an architecture described more fully in section 5.8. Threading does not offer true concurrency[3], but does allow Python to distribute computational

[3] See David Beazley's 'Understanding the Global Interpreter Lock' and associated visualizations.

time between operations so that, for example, waiting for an event does not block the rest of the program, and events are not missed because the program is busy (Figure 4.2).

Critical operations that are computationally intensive or cannot be interrupted are given their own dedicated **processes**. Linux allows individual cores of a processor to be reserved for single processes, so individual Raspberry Pis are capable of running four truly parallel processing streams. For example, all Raspberry Pis in an Autopilot swarm create a messaging client to handle communication between devices which runs on its own processor core so no messages are missed. Similarly, if an experiment requires sound delivery, a realtime sound engine in a separate process (Figure 4.3) also runs on its own core.

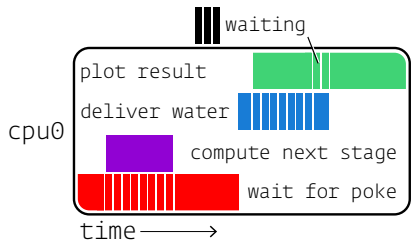


Figure 4.2: A multi-threaded program divides computation time of a single process and cpu core across multiple operations so that, for example, waiting for input doesn't block other operations.

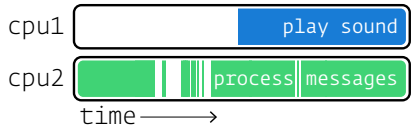


Figure 4.3: A multi-process program is truly concurrent, allowing multiple cpu cores to operate in parallel.

Since even moderately complex experiments can consume more resources than are available on a single processor, the topmost layer of concurrency in Autopilot is to use additional **computers**. Autopilot uses the Raspberry Pi as a low-cost hardware controller, but only its GPIO control system is unique to them: the rest of the code can be used on any type of computer, so computationally expensive or GPU-intensive operations can be offloaded to any number of high performance machines. Computers divide labor *autonomously* (see 4.2 and 5.7), so for example one computer running a task can send and receive messages from another running the GUI and plots, but does not *depend* on that input as it would in a system that couples a microcontroller with a managing computer. The ability to coordinate multiple, autonomous computers with heterogeneous responsibilities and capabilities in a shared task is Autopilot's definitive design decision.

Leveraging Low-Level Libraries

Autopilot uses Python as a "glue" language, where it wraps and coordinates faster low-level compiled code[204]. Performance-critical components of Autopilot are thin wrappers around fast C libraries (Table 4.1). As Autopilot's API matures, we intend to replace any performance-limiting Python code like its sound server and networking operations with compiled code exposed to python with tools like the C Foreign Function Interface (CFFI).

Table 4.1: A few libraries Autopilot uses

jack	realtime audio
pigpio	GPIO control
ZeroMQ	networking
Qt	GUI

Since Autopilot coordinates its low-level components in parallel rather putting everything inside one "main loop," Autopilot actually has *better* temporal resolution than single-threaded systems like Bpod or pyControl, despite the realtime nature of their dedicated processors (Table 4.2).

Table 4.2: Using pigpio as a dedicated I/O process gives autopilot greater measurement precision

	Precision
Autopilot (pigpio)	$5\mu s$
Bpod	$100\mu s$
pyControl	$1000\mu s$

Caching

Finite-state machines are only aware of the current state and the events that transition it to future states. They are thus incapable of exploiting the often predictable structure of behavioral tasks to precompute future states and precache stimuli. Further, to change task parameters between trials (eg. changing the rewarded side in a two-alternative forced-choice task), state machines need to be fully reconstructed and reuploaded to the device that runs them each time.

Autopilot precomputes and caches as much as possible. Rather than wait "inside" a state, Autopilot prepares each of the next possible events and saves them for immediate execution when the appropriate trigger is received. Static stimuli are prepared

once at the beginning of a behavioral session and stored in memory. Before their presentation, they are buffered to minimize latency.

By providing full low-level documentation, we let researchers choose the balance between ease of use and performance themselves: it's possible to just call a sound's `play()` method, explicitly buffer it with its `buffer()` method, or generate samples on the fly with its `play_continuous()` method. Similarly, messages can be sent with a networking node's `send()` method, or prepared beforehand by explicitly making a `Message` and calling its `serialize()` method.

Autopilot's efficient design lets it access the best of both worlds—the speed and responsiveness of compiled code on dedicated microprocessors and the accessibility and flexibility of interpreted code.

4.2 Flexibility

Single-language

Behavior software that uses dedicated microprocessors must have some routine for compiling the high-level abstraction of the experiment into machine code. This gives those systems a theoretical advantage in processing speed, but the compiler becomes the bottleneck of complexity: only those things that can be compiled can be included in the experiment. This may in part contribute to the ubiquity of state-machine formalisms in behavior software.

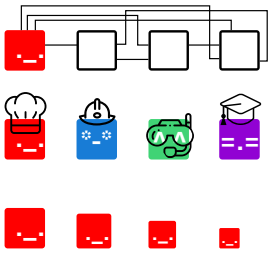
Because Python is used throughout the system, extending Autopilot's functionality is straightforward. Task design (see section 5.3) is effectively arbitrary—anything that can be expressed in Python is a valid task. This also allows Autopilot to easily be extended to make use of external libraries (eg. our integration with DeepLabCut-Live[205] and our planned integration with OpenEphys).

Modularity

Although Autopilot deeply integrates with the Raspberry Pi's hardware, we have also worked to make its components modular. There is a tension between providing a full-featured behavioral system and the flexibility of its components — as additional features are added to a system, they can constrain the functionality of existing components that they rely on. To address this tension, we have continuously worked to decouple Autopilot into subcomponents with clear inheritance hierarchies and APIs that can used quasi-independently.

Modularity has 3 primary advantages:

1. **Modularity makes code more flexible** by reducing the constraints imposed by unstructured code interdependencies
2. **Modularity makes code more intelligible** by logically distributing tasks to discrete classes
3. **Modularity reduces effort-duplication** by allowing multiple, similar classes to be created with inheritance rather than copying and pasting.



There is no such thing as "incompatible hardware" with Autopilot because the classes that control hardware are independent from the code that provides other core functionality. In systems without modular design, hardware implementation is spread across the codebase. For example to add a new type of hardware output to a Bpod system, one would need to write new firmware for it in C (eg. the valve driver module), modify Bpod's existing firmware, hunt through the code to modify how states are added and state machines are assembled, add its controls explicitly to the GUI, and so on.

Tasks specify what type of hardware is needed to run them, but are agnostic about the way the hardware is implemented, making their descriptions more portable. Tasks that have the same structure but differ in hardware (eg. a freely moving two-alternative forced choice task in which a mouse visits several IR sensors, or a head-fixed two-alternative forced choice task in which a mouse runs on a wheel to indicate its choice) can be implemented by a trivial subclass that modifies the hardware description rather than completely rewriting the task.

Plugins & Code Transparency

We call Autopilot a software framework because in addition to providing classes and methods to run experiments out of the box, it also provides explicit structure that scaffolds any additional code that is needed by the user. Our goal is to clearly articulate in the documentation how modules should interact so that anyone can write code that works on any apparatus.

As groupware intended to be used differently by lab members with different responsibilities, Autopilot is designed for users with a range of programming expertise, from those who only want to interact with a GUI, to those who wish to fundamentally rewrite core operations for their particular experiment. As such, it is extensively documented: this paper provides a high-level introduction to its design and structure, its user guide describes how to use the program and provides examples, and its API-level documentation describes in granular detail how the program actually works[4]. Nothing is "off-limits" to the user—there isn't any hidden, undocumented hardware code behind the curtain[5]. We want users to be able to understand how and why everything works the way it does so that Autopilot can be adapted and expanded to any use-case.

A broader goal of Autopilot is to build a library of flexible task prototypes that can be tweaked and adapted, hopefully reducing the number of times the wheel is reinvented. We have attempted to nudge users to write reusable tasks by designing Autopilot such that rather than writing tasks as local unstructured scripts, they use its plugin system that scaffolds development by extending any of its basic types. Plugins are registered using a form in the Autopilot Wiki which makes them available to anyone while also embedding them in a semantically annotated information system that allows giving explicit credit to contributors, programmatically linking to any derivative publications that use the plugin, and further documentation of any tasks, hardware, or other extensions included within the plugin. Inheriting from parent classes give plugins structure and a set of basic features[6] while also being maximally permissive — anything can be overridden and modified.

[4] The user guide and API documentation are available at `docs.auto-pi-lot.com`

[5] For readability of the docs, we omit generating HTML documentation for some private methods and functions, but they are documented in the source and their function is made clear from their context and the documentation of public methods.

[6] Like inheriting from the `GPIO` class gives GPIO plugins a systematic means of interacting with the underlying pigpiod daemon.

Message Handling

Modular software needs a well-defined protocol to communicate between modules, and Autopilot's is heavily influenced by the concurrency philosophy[7] of ZeroMQ[206]. All communication between computers and modules happens with ZeroMQ messages, and handling those messages is the main way that Autopilot handles events. A key design principle is that Autopilot components should not "share state"—they can communicate, but they are not *dependent* on one another. While this may seem like a trivial detail, having networking and message-handling at its core has three advantages that make Autopilot a fundamental departure from previous behavioral software.

[7] "ZeroMQ [...] has a subversive effect on how you develop network-capable applications. [...] message processing rapidly becomes the central loop, and your application soon breaks down into a set of message processing tasks."

"If there's one lesson we've learned from 30+ years of concurrent programming, it is: *just don't share state.*"

-The ZeroMQ Guide

First, new software modules can be added to any system by simply dropping in a standalone networking object. There is no need to dramatically reorganize existing code to make room for new functionality. Instead new modules can receive, process, and send information by just connecting to another module in the swarm. For example, each plot opens a network connection to stream incoming task data independently from the stream that is saving the data.

Second, Autopilot can be made to interact with other software libraries that use ZeroMQ. For example, The OpenEphys GUI for electrophysiology can send and receive ZMQ messages to execute actions such as starting or stopping recordings. Interaction with other software is also useful in the case that some expensive computation needs to happen mid-task. For example, one could send frames captured from a video camera on a Raspberry Pi to a GPU computing cluster for tracking the position of the animal. Since ZeroMQ messages are just TCP packets it is also possible to communicate over the internet for remote control or to communicate with a data server.

Third, making every component network-capable allows tasks to be distributed over multiple Raspberry Pis. Chaining multiple Pis distributes the computational load, allowing, for example, one Raspberry Pi to record and process video while another runs a sound server and delivers rewards. Autopilot expands with the complexity of your task, simultaneously eliminating limitations on quantity of hardware peripherals while ensuring latency is minimal. More interestingly, distributing tasks allows the arbitrary construction of what we call "behavioral topologies," which we describe in section 5.7.

4.3 Reproducibility

We take a broad view on reproducibility: including not only the ability to share data and recreate experiments, but also integrating into a broader ecosystem of tools that reduces labor duplication and encourages sharing and organizing technical knowledge. For us, reproducibility means building a set of tools that make every experiment and every technique available to anyone, anywhere.

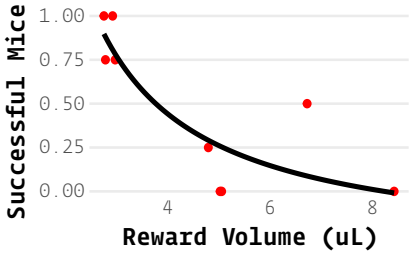


Figure 4.4: "Minor" details have major effects. Proportion of mice (each point, n=4) that were successful learning the first stage of the speech task described in [136] across 10 behavior boxes with variable reward sizes. A $2\mu L$ difference in reward size had a surprisingly large effect on success rate.

Standardized task descriptions

The implementation and fine details of a behavioral experiment matter. Seemingly trivial details like milliseconds of delay between trial phases and microliters of reward volume can be the difference between a successful and unsuccessful task (Fig-

ure 4.4). *Reporting* those details can thus be the difference between a reproducible and unreproducible result. Researchers also often use "auxiliary" logic in tasks—such as methods for correcting response bias—that are never completely neutral for the interpretation of results. These too can be easily omitted due to brevity or memory in plain-English descriptions of a task, such as those found in Methods sections. Even if all details of an experiment were faithfully reported, the balkanization of behavioral software into systems peculiar to each lab (or even to individuals within a lab) makes actually performing a replication of a behavior result expensive and technically challenging. Widespread use of experimental tools that are not explicitly designed to preserve every detail of their operation presents a formidable barrier to rigorous and reproducible science[188].

Autopilot splits experiments into a) the **code** that runs the experiment, which is intended to be standardized and shared across implementations, and b) the **parameters** (Figure 4.5) that define your particular experiment and system configuration. For example, two-alternative forced choice tasks have a shared structure regardless of the stimulus modality, but only your task plays pitch-shifted national anthems. This division of labor, combined with Autopilot's structured plugin system, help avoid the ubiquitous problem of rig-specific code and hard-coded variables making experimental code only useful on the single rig it was designed for — enabling the possibility of a shared library of tasks as described in section 4.2

The practice of reporting exactly the parameter description used by the software to run the experiment removes any chance for incompleteness in reporting. Because all task parameters are included in the produced data files, tasks are fully portable and can be reimplemented exactly by anyone that has comparable hardware to yours.

```
{
"step_name"    : "tone_discrim",
"task_type"    : "2AFC",
"bias_mode"    : 0,
"punish_sound" : false,
"stim" : {
  "sounds" : {
    "L": {
      "duration"  : 100,
      "frequency" : 10000,
      "type"      : "tone",
      "amplitude" : 0.01},
    "R": {"...":"..."}}},
"reward": {
  "type"     : "volume",
  "volume"   : 20},
"graduation" : {
    "type"      : "accuracy",
    "threshold" : 0.75,
    "window"    : 400},
}
```

Figure 4.5: Task parameters are stored as portable JSON, formatting has been abbreviated for clarity.

Self-Documenting Data

A major goal of the open science movement is to normalize publishing well documented and clearly formatted data alongside every paper. Typically, data are acquired and stored in formats that are lab-idiosyncratic or ad-hoc, which, over time, sprout entire software libraries needed just to clean and analyze it. Idiosyncratic data formats hinder collaboration within and between labs as the same cleaning and analysis operations gain multiple, mututally incompatible implementations, duplicating labor and multiplying opportunities for difficult to diagnose bugs. Over time these data formats and their associated analysis libraries can mutate and become incompatible with prior versions, rendering years of work inaccessible or uninterpretable. In one worst-case scenario, the cleaning process unearths some critically missing information about the experiment, requiring awkward caveats in the Methods section or months of extra work redoing it. In another, the missing information or bugs in analysis code are never discovered, polluting scientific literature with inaccuracies.

The best way to make data publishable is to avoid cleaning data altogether and *design good data hygiene practices into the data acquisition process.* Autopilot automatically stores all the information required to fully reconstruct an experiment, including any changes in task parameters or code version that happen throughout training as the task is refined.

Autopilot data is stored in HDF5 files, a hierarchical, high-performance file format. HDF5 files support metadata throughout the file hierarchy, allowing annotations

```
root
├── current_task
├── data
│   └── task_name
│       ├── S00_free_water
│       │   └── trial_data
│       └── S01_tone_discrim
│           └── trial_data
├── history
│   ├── git_hashes
│   ├── parameter_history
│   ├── past_protocols
│   │   └── old_task
│   └── weights
└── info
    ├── animal_id
    ├── birth_date
    ├── genotype
    └── etc_additional_info
```

Figure 4.6: Example data structure. All information necessary to reconstruct an experiment is automatically stored in a human-readable HDF5 file.

to natively accompany data. Because HDF5 files can store nearly all commonly used data types, data from all collection modalities—trialwise behavioral data, continuous electrophysiological data, imaging data, etc.—can be stored together from the time of its acquisition. Data is always stored with the full conditions of its collection, and is ready to analyze and publish immediately (Figure 4.6). No Autopilot-specific scripts are needed to import data into your analysis tool of choice—anything that can read HDF5 files can read Autopilot data[8].

[8] Though our `Subject` class provides a simplified interface to access and manipulate Autopilot data

As of v0.5.0, we have built a formal data modeling system into Autopilot, allowing for unified declaration of data for experimental subjects, task parameters, and resulting data with verifiable typing and human-readable annotations. These abstract data models can be used with multiple storage interfaces, paving the way for export to, for example, the Neurodata Without Borders standard[207], further enabling Autopilot data to be immediately incorporated into existing processing pipelines (see section 5.2).

Testing & Continuous Integration

Open-source scientific software does away with prior limitations to access and inspection imposed by proprietary tools. It also exposes the research process to bugs in software written by semi-amateurs that can yield errors in the resulting data, analysis, and interpretation[208, 209, 210, 211]. Autopilot tries to bring best practices in software development to experimental software, including a set of automated tests for continuous integration.

We are still formalizing our contribution process, and our tests are still far from achieving full coverage[9], but we currently require tests and documentation for all new code added to the library. Writing good tests is hard, and we are in the process of building a set of hardware simulators and test fixtures to ease contribution.

[9] Coverage statistics for Autopilot are available on coveralls.io at https://coveralls.io/github/auto-pi-lot/autopilot

```python
def test_set_gpio():
    """
    The `set` method of a Digital_Out
    object sets the pin state
    """
    pin = Digital_Out(pin=17)

    # Turn GPIO pin on
    pin.set(True)
    assert pin.state == True

    # Turn GPIO pin off
    pin.set(False)
    assert pin.state == False
```

Figure 4.7: A test like `test_set_gpio` is a provable statement about the functionality of a program, in this case that "the `Digital_Out.set()` method sets the state of a GPIO pin."

Tests are effectively provable statements about how a program functions (Figure 4.7), which are particularly important for a library that aspires to be baseline lab infrastructure like Autopilot. Tests make it possible to use and contribute to the library with confidence: all tests are run on every commit, making it possible to determine if some new contribution breaks existing code without manually reading and testing every line. As we work to complete our test coverage, we hope to provide researchers with a tool that they can trust and elevates the verifiability of scientific results at large.

Expense

Autopilot is an order of magnitude less expensive than comparable behavioral systems (Table 4.3). We think the expense of a system is important for two reasons: scientific equity and statistical power.

The distribution of scientific funding is highly skewed, with a large proportion of research funding concentrated in relatively few labs[212]. Lower research costs benefit all scientists, but lower instrumentation costs directly increase the accessibility of state-of-the-art experiments to labs with less funding. Since well-funded labs also tend to be concentrated at a few (well-funded) institutions, lower research costs also broaden the base of scientists outside traditional research institutions that can stay at the cutting edge[213, 214, 215].

Neuroscience also stands to benefit from the lessons learned from the replication crisis in Psychology[217]. In neuroscience, underpowered experiments are the rule, rather than the exception[218]. Statistical power in neuroscience is arguably even worse than it appears, because large numbers of observations (eg. neural recordings) from a small number of animals are typically pooled, ignoring the nested structure of observations collected within individual animals. Increasing the number of cells recorded from a small number of animals dramatically increases the likelihood of Type I errors (Figure 4.8)—indeed, for values of within-animal correlation typical of neuroscientific data, high numbers of observations make Type I errors more likely than not[216]. For this reason, perhaps paradoxically, recent technical advances in multiphoton imaging and silicon-probe recordings will actually make statistical rigor in neuroscience *worse* if we don't use analyses that account for the multilevel structure of the data and correspondingly record from the increased number of animals that they require.

Although the expense of multi-photon imaging and high-density electrophysiology will always impose an experimental bottleneck, behavioral training time is often the greater determinant of study sample size. Typical behavioral experiments require daily training sessions often carried out over weeks and months, while far fewer imaging or electrophysiology sessions are carried out per animal. Training large cohorts of animals in parallel is thus the necessary basis of a well-powered imaging or electrophysiology experiment.

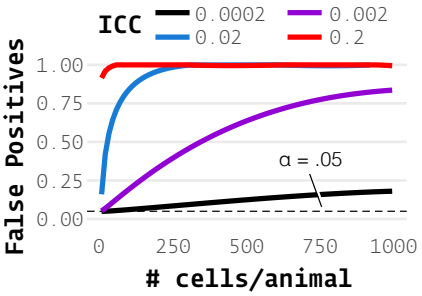


Figure 4.8: When comparing a value across groups, eg. a genetic knockout vs. wildtype, even a modest intra-animal (or, more generally, intra-cluster) correlation (ICC) causes the false positive rate to be far above the nominal $\alpha = 0.05$. Shown are false positive rates for simulated data with various numbers of "cells" recorded for comparisons between two groups of 5 animals each with a real effect size of 0. We note that 741 simultaneously recorded cells were reported in [181] and a mean ICC of 0.19 across 18 neuroscientific datasets was reported in [216]

	Autopilot	pyControl	Bpod
Behavior CPU	$45	$270	$925
Nosepoke (3x)	$216	$369	$810
Total for One	**$261**	**$639**	**$1735**
Five Systems	$1305	$3195	$8675
Host CPU(s)	$1000	$1000	$5000
Total for Five	**$2305**	**$4195**	**$13625**
Total for Ten	**$3610**	**$8390**	**$27350**

Table 4.3: Cost for Basic 2AFC System
"Nosepoke" includes a solenoid valve, IR sensor, water tube, LED, housing, and any necessary driver PCBs. For PyControl and Autopilot, we included the cost of one Lee LHDA0531115H solenoid valve per nosepoke ($63.35). For PyControl, we estimated a typical USB hub with 5 ports to control 5 pyControl systems from one computer. We note that the Bpod and PyControl systems both include cost of assembly for the control CPUs and nosepokes, but also that Autopilot does not require assembly for its control CPU and its default nosepoke is a snap-together 3D printed part and PCB without surface mounted components that can be assembled by an amateur in roughly half an hour.

5

Program Structure

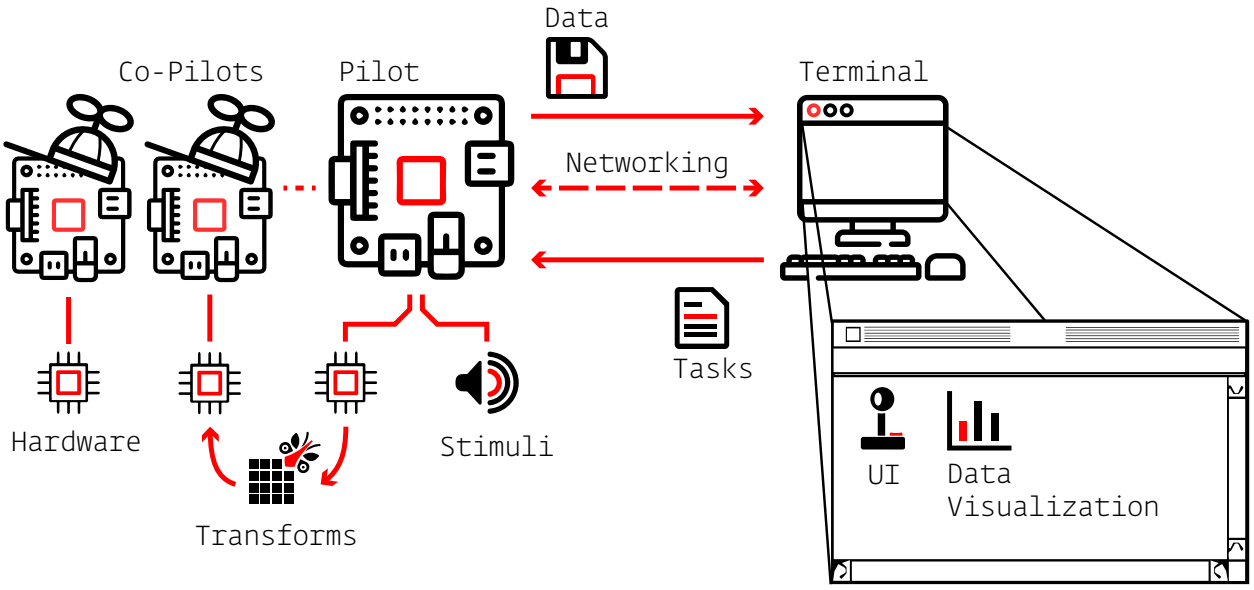


Figure 5.1: Overview of major Autopilot components

Autopilot consists of software and hardware modules configured to create a behavioral topology. Independent agents linked by flexible networking objects fill different roles within a topology, such as hosting the user interface, controlling hardware, transforming incoming and outgoing data, or delivering stimuli. This infrastructure is ultimately organized to perform a behavioral task.

5.1 Directory Structure

On setup, Autopilot creates a user directory that contains all local files that define its operation (Figure 5.2). The subdirectories include:

- **`calibration`** — Calibration for hardware objects like audio or solenoids that, for example, map opening durations to volumes of liquids dispensed
- **`data`** — Data for experimental subjects
- **`launch_autopilot.sh`** — Launch script that includes launching external processed like the jack audio daemon (will be removed and integrated into a more formal agent structure in future versions)
- **`logs`** — Every Autopilot object is capable of full debug logging, neatly formatted by object type and instance ID and grouped within module-level logging files. Logs are both written to disk, and output to `stderr` using the rich logging handler for clean and readable inspection during program operation (Figure 5.3).

```
./autopilot
├── calibration
├── data
│   ├── subject_1.h5
│   └── subject_2.h5
├── launch_autopilot.sh
├── logs
│   ├── core.terminal.log
│   └── plugins.my_plugin.log
├── pilot_db.json
├── plugins
│   └── my_plugin
│       └── my_task.py
├── prefs.json
├── protocols
│   ├── 2afc_easy.json
│   └── 2afc_hard.json
└── sounds
```

Figure 5.2: Example user directory structure, typically in ~/autopilot.

Logs can be parsed back into python objects to make it straightforward to diagnose problems or recover data in the case of an error.

```
[22-03-17T20:10:39] INFO     [core.terminal] parent, module-level logger created: core.terminal       loggers.py:159
                    INFO     [core.terminal.Terminal] Logger created: core.terminal.Terminal          loggers.py:163
                    INFO     [core.terminal.Terminal] successfully loaded pilot_db.json file from    terminal.py:437
                             /Users/jonny/autopilot/pilot_db.json
                    DEBUG    [core.terminal.Terminal] OrderedDict([('added_pilot',                   terminal.py:438
                                        OrderedDict([('subjects', ['myid']),
                                                     ('ip', ''),
                                                     ('prefs', OrderedDict())])),
                                       ('platform',
                                        OrderedDict([('subjects', ['test_nafc']),
                                                     ('ip', '192.168.0.101'),
                                                     ('prefs', OrderedDict())]))])
[22-03-17T20:10:39] INFO     [core.gui] parent, module-level logger created: core.gui                 loggers.py:159
                    INFO     [core.gui.Control_Panel] Logger created: core.gui.Control_Panel          loggers.py:163
[22-03-17T20:10:39] INFO     [core.plots] parent, module-level logger created: core.plots             loggers.py:159
                    INFO     [core.plots.Plot_Widget] Logger created: core.plots.Plot_Widget          loggers.py:163
                    INFO     [core.plots.Plot] Logger created: core.plots.Plot                        loggers.py:163
```

Figure 5.3: Logs printed to `stderr` are formatted and colorized by the rich logging handler. Logfiles are created by module, and log entries are identified by the individual objects instantiated from them. Logfiles are rotated and size-limited for configurable backups.

- **`pilot_db.json`** — A .json file that stores information about associated Pilots, including the contents of their prefs files, which hash/version of Autopilot they are running, and any Subjects that are associated with them.
- **`plugins`** — Plugins, which are any Python files that contain subclasses of Autopilot objects, that are automatically made available by Autopilot's registry system (eg. `autopilot.get('hardware', 'My_Hardware')` would retrieve a custom hardware object). Plugins can be documented and made available to other Autopilot users by registering them on the wiki
- **`prefs.json`** — Configuration options for this particular Autopilot instance, including configurations of local hardware objects, audio output, etc. In the future this will likely be broken into multiple files for different kinds of preferences[1].
- **`protocols`** — Protocols, which consist of parameterizations of individual Tasks as well as criteria for graduating betwewen them. These are also stored in individual subject data files, and updated whenever the source protocol files change.
- **`sounds`** — Any sound files that are requested by the File sound class.

[1] with care for backwards compatibility

5.2 Data

As of v0.5.0, Autopilot uses pydantic to create explicitly typed and schematized data models. Submodules include `data` abstract `modeling` tools that define base model types like `Tables`, `Groups`, and sets of `Attributes`. These base modeling classes are then built into a few core data models like subject `Biography` information, `Protocol` declaration, and the `Subject` data model itself that combines them. Modeling classes then have multiple `interfaces` that can be used to create equivalent objects in other formats, like pytables for hdf5 storage, pandas dataframes for analysis, or exported to Neurodata Without Borders.

For example, consider a simplified version of the `Biography` model:

Listing 1: **data - Biography**

```
from autopilot.data.modeling import Data, Attributes, Field
from typing import Optional, Union
from datetime import datetime, timedelta

class Enclosure(Data):
    """Where does the subject work?"""
    box:  Optional[Union[str, int]] = Field(
        default=None,
        description="The box this Subject is run in")
    room: Optional[Union[str, int]] = Field(
        default=None,
        description="The room number that the animal is run in")

class Biography(Attributes):
    """Biography of an Experimental Subject"""
    id:  str = Field(...
        description="The indentifying name of this subject.")
    dob: datetime = Field(...
        description="The Subject's date of birth")
    enclosure: Optional[Enclosure] = None

    @property
    def age(self) -> timedelta:
        """Difference between now and :attr:`.dob`"""
        return datetime.now() - self.dob
```

Data models use builtin Python type hints. Type hints are colon delimited annotations like `x:int` that indicate the type (integer, string, etc.) of the variable. Though typically Python does not, Pydantic both validates that a type matches its hint and coerces it to the correct type if possible.

The `Union` type means that a field can be one of several possible types, in this case the box can be identified with either a string or integer.

Optional fields can have default fields, either a single value like **`None`** or a function that computes a default value like the current date.

Descriptions are stored in the data model schema to make shared data self-documenting, and also used by GUI widgets for tooltips that clarify what fields mean.

The class that our model inherits from indicates how Autopilot should treat it in a given storage interface. For HDF5 files, subclasses of the Attributes class are stored as node metadata, while subclasses of Table make tables.

Autopilot's format interfaces define mappings from nonstandard types to types supported by the format. For HDF5 files, datetime objects are converted to ISO 8601 formatted strings

Data models can be recursive, or use other models as types for their own fields. In this case the subject's Enclosure can be optionally specified in its Biography.

Properties are model fields that are automatically computed based on the values of other fields. The age of the animal can be accessed like a normal instance attribute that returns a timedelta object.

A new subject could then be created with a biography like this, storing it in the HDF5 file and made accessible through the Subject interface:

Listing 2: **data - New Subject**

```
from autopilot.data import Subject
from autopilot.data.models import Biography, Enclosure
bio = Biography(
    id="my_subject",
    dob="2022-01-01T00:00:00",
    enclosure=Enclosure(box=100, room="Building 200")
)
sub = Subject.new(bio)
assert sub.info == bio
```

The `Subject` class is the primary means by which Autopilot stores, organizes, and interacts with data.

Several basic models are built into Autopilot, and in future versions it will be possible to extend and replace these models with plugins, making storage formats fully customizable while still being explicit and understandable.

If we don't give the type specified in the model, it will try and coerce it to the correct type and raise an error if it can't.

An **`assert`**ion declares that some logical statement is **`True`** and raises an exception if it isn't. Autopilot's unit tests ensure that subject data can be stored and retrieved without losing information or changing types.

The models are declared using a combination of python type hints and `Field` objects that provide defaults and descriptions. Because these models can be recursive, as in the case of using the `Enclosure` model as a type within the `Biography` model, we can build expressive, flexible, but still strict representations of complex data.

Out of the box, pydantic models can create explicit and interoperable schemas in JSON Schema and OpenAPI formats, and Autopilot extends them with additional interfaces and representations. Autopilot can create a GUI form for filling in fields for models, for example, to create a new Subject or declare parameters for a task (Figure 5.4). `Attribute` models that consist of scalar key-value pairs can be reliably stored and retrieved from metadata attribute sets in HDF5 groups, but Autopilot knows that `Table` models should be created as HDF5 tables as they will have multiple values for each field. An additional `Trial_Data` class that inherits from `Table` can be exported to NWB trial data, and the `Subject.get_trial_data` method uses the model to load trial data and convert it to a correctly typed pandas[219] DataFrame.

Though the data modeling system is new in v0.5.0[2], we have laid the groundwork for Autopilot's plugin system to allow researchers to declare custom schema for all data produced by Autopilot, and to preserve both interoperability and reproducibility by combining them with datasets potentially produced by multiple incompatible tools (see Section 7.4).

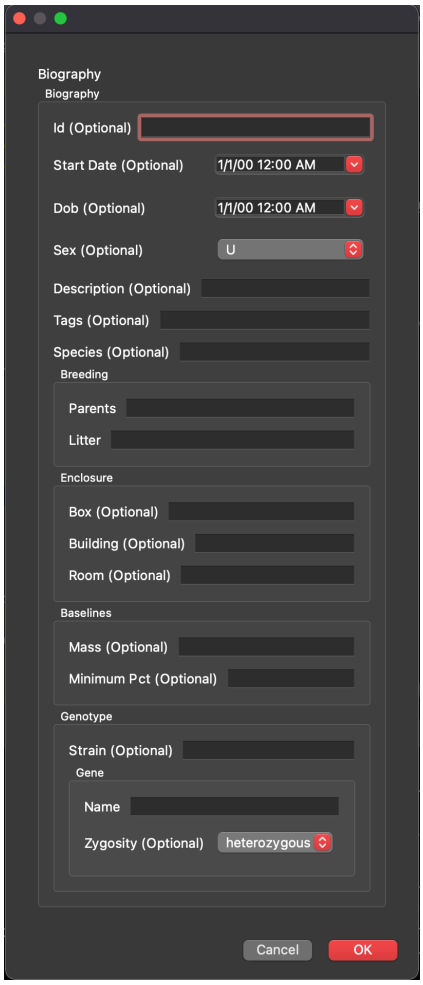


Figure 5.4: An Autopilot Data model can automatically generate a GUI form to fill in its properties, in this example to define a new experimental Subject's biography.

[2] Released as an alpha version at the time of writing

5.3 *Tasks*

Behavioral experiments in Autopilot are centered around **tasks**. Tasks are Python classes that describe the parameters, coordinate the hardware, and perform the logic of the experiment. Tasks may consist of one or multiple **stages** like a stimulus presentation or response event, completion of which constitutes a **trial** (Figure 5.5). Stages are analogous to states in the finite state machine formalism.

Multiple tasks are combined to make **protocols**, in which animals move between tasks according to "graduation" criteria like accuracy or number of trials. Training an animal to perform a task typically requires some period of shaping where they are familiarized to the apparatus and the structure of the task. For example, to teach animals about the availability of water from "nosepoke" sensors, we typically begin with a "free water" task that simply gives them water for poking their nose in them. Having a structured protocol system prevents shaping from relying on intuition or ad hoc criteria.

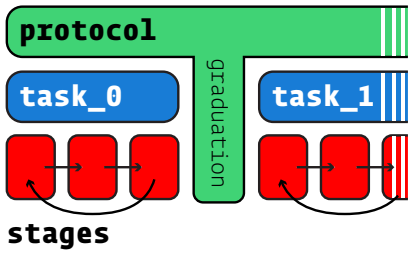


Figure 5.5: Protocols consist of one or multiple tasks, tasks consist of one or multiple stages. Completion of all of a task's stages constitutes a trial, and meeting some graduation criterion like accuracy progresses a subject between tasks.

Task Components

The following is a basic two-alternative choice (2AFC) task—a sound is played and an animal is rewarded for poking its nose in a designated target nosepoke. While simple, it is included here in full to show how one can program a task, including an explicit data and plotting structure, in roughly 60 lines of generously spaced Python.

Every task begins by describing four elements:

1) the task's parameters, 2) the data that will be collected, 3) how to plot the data, and 4) the hardware that is needed to run the task.

Listing 3: task - parameters

```
class Nafc(Task):
    class Params(Task_Params):
        stim: Sounds = Field(...,
            description = "Sound Stimuli")
        reward: units.mL = Field(...,
            description = "Reward Volume (mL)"
        )

    class TrialData(Trial_Data):
        target: Side = Field(...,
            description="Side (L, R) of the correct response")
        correct: bool = Field(...,
            description="Response matched target")

    PLOT = {}
    PLOT['data']   =  {'target'  : 'point',
                       'correct' : 'rollmean'},
    # n trials to roll window over
    PLOT['params'] = {'roll_window' : 50}

    HARDWARE = {
        'POKES':{
            'L': 'Digital_In',
            'R': 'Digital_In'
        },
        'PORTS':{
            'C': 'Solenoid',
        }
    }
```

1) A `Task_Params` model defines what parameters are needed to run the task.

We use `Field` objects as in listing 1, and can also use some special types like `Sounds` to declare complex parameters

`units` work like numbers but avoid ambiguity, so eg. the `Solenoid` class below knows this is a volume, rather than a duration

2) A `Trial_Data` model defines what data will be returned from the task.

3) A `PLOT` dictionary maps the data output to graphical elements in the GUI. (In future versions this will be incorporated into the `Fields` of `TrialData`)

4) A `HARDWARE` dictionary that describes what hardware will be needed to run the task.

The specific implementation of the hardware (eg. where it is connected, how to interact with it) is independent of the task. The task just knows about a `PORT` named `'C'` that is a `Solenoid`.

Created tasks receive some common methods, like input/trigger handling and networking, from an inherited metaclass. Python inheritance can also be used to make small alterations to existing tasks[3] rather than rewriting the whole thing. The GUI will use the `Params` model and the `PLOT` dictionary to generate forms for parameterizing the task within a protocol and display the data as it is collected. The `Subject` class will use the `TrialData` model to create HDF5 tables to store the data, and the `Task` metaclass will instantiate the described `HARDWARE` objects from their system-specific configuration in the `prefs.json` file so they are available in the rest of the class like `self.hardware['POKES']['L'].state`

[3] An example of subclassing a generic 'Task' class is included in Autopilot's user guide

Stage Methods

The logic of tasks is described in one or a series of methods (stages). The order of stages can be cyclical, as in this example, or can have arbitrary logic governing the transition between stages.

Listing 4: task - methods

```
30  def __init__(self, params:'Nafc.Params'):
31      self.stim_mgr = Stim_Manager(params.stim)
32      self.reward   = Reward_Manager(params.reward)
33
34      stage_list  = [self.discrim, self.reinforcement]
35      self.stages = itertools.cycle(stage_list)
36
37      self.init_hardware()
38      next(self.stages)()
39
40  def discrim(self):
41      target, wrong, stim = self.stim_mgr.next()
42      self.target = target
43
44      self.triggers[target] = [
45          self.hardware['PORTS']['C'].open,
46          lambda: next(self.stages)()]
47      self.triggers[wrong] = lambda: next(self.stages)()
48
49      self.node.send('DATA', {'target':target})
50
51      stim.play()
52
53  def reinforcement(self, response):
54      if response == self.target:
55          self.node.send('DATA', {'correct':True})
56      else:
57          self.node.send('DATA', {'correct':False})
58
59      next(self.stages)()
```

In Python, `def` defines new methods. The `__init__` method is called when a new object is initialized

Managers control stimulus and reward delivery, so users can, for example, continually synthesize new stimuli or implement adaptive rewards

Stages are combined into an object that (in this case) continually cycles through them when its `next()` method is called.

This starts the task by retrieving the first stage and then calling it.

The stimulus manager returns which port will be the target and the sound to be played.

A sequence of triggers is set: if the target port is poked, a reward will be delivered and the next stage will be called. A `lambda` function indicates not to call the method *now*, but only when triggered.

The task has a networking object that asynchronously streams data back to the user-facing terminal

In this example, the response port is passed from the trigger handling function. If it matches the stored target variable, the animal answered correctly.

Finally, the task is repeated by calling the next stage.

Autopilot is not prescriptive about how tasks are written. The same task could have two separate methods for correct and incorrect answers rather than a single reinforcement method, or only a single stage that blocks the program while it waits for a response.

Publishing data from this task requires no additional effort: a hash that uniquely identifies the code version (as well as any local changes) is automatically stored at the time of collection, as is a JSON-serialized version of the parameter model (Figure 5.6). If this task was incorporated into the central task library, anyone using Autopilot would be able to exactly replicate the experiment from the published data.

```
{
"step_name": "Simple 2AFC",
"stim" : {
  "sounds" : {
    "L": {
      "type"      : "tone",
      "frequency" : 4000},
    "R": {
      "type"      : "tone",
      "frequency" : 8000}
  }
},
"reward": 10
}
```

Figure 5.6: Simplified example of parameters for the above task

The limitations of finite state machines

The 2AFC task described above could be easily implemented in a finite-state machine. However, the difficulty of programming a finite-state machine is subject to combinatoric explosion with more complex tasks. Specifically, finite-state machines can't handle any task that requires any notion of "state history."

As an example, consider a maze-based task. In this task, the animal has to learn a particular route through a maze—it is not enough to reach the endpoint, but the animal has to follow a specific path to reach it (Figure 5.7). The arena is equipped with an actimeter that detects when the animal enters each area.

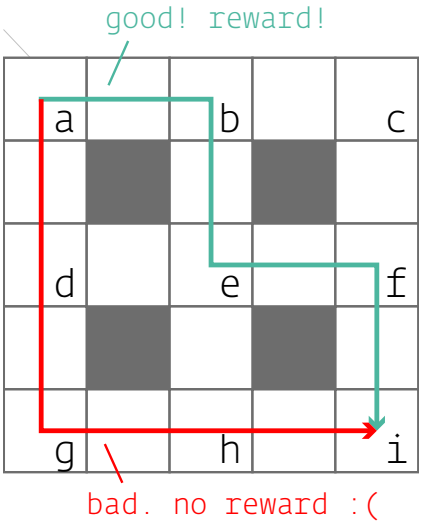


Figure 5.7: The subject must reach point `i` but only via the correct (green) path.

In Autopilot, we would define a hardware object that logs positions from the actimeter with a `store_position()` method. If the animal has entered the target position ("i" in this example), a `task_trigger()` that advances the task stage is called. The following code is incomplete, but illustrates the principle.

Listing 5: **maze - hardware**

```
1  class Actimeter(Hardware):
2      def __init__(self):
3          # ... some code to access the hardware ...
4          self.positions = []
5          self.target_position = "i"
6
7      def store_position(self, position):
8          self.positions.append(position)
9
10         if position == self.target_position:
11             self.finished_cb(self.positions)
12             self.positions = []
```

See line 18 below

The task follows, with parameters and network methods for sending data omitted for clarity.

Listing 6: **maze - task**

```
13 class Maze(Task):
14     def __init__(self):
15         self.target_path = ['a', 'b', 'e', 'f', 'i']
16
17         self.actimeter = Actimeter()
18         self.actimeter.finished_cb = self.finished
19
20     def finished(self, positions):
21         if positions == self.target_path:
22             self.reward()
```

The actimeter is given a reference to the Maze task's `finished()` method, which it calls when the target position is reached

The sequence of `positions` is compared to the `target_path` with ==. If they match, the subject is rewarded!

How would such a task be programmed in a finite-state machine formalism? Since the path matters, each "state" needs to consist of the current position and all the positions before it. But, since the animal can double back and have arbitrarily many state transitions before reaching the target corner, this task is impossible to represent with a finite-state machine, as a full representation would necessitate infinitely many states (this is one example of the *pumping lemma*, see [220]).

Even if we dramatically simplify the task by 1) assuming the animal never turns back and visits a space twice, and 2) only considering paths that are less than or equal to the length of the correct path, the finite state machine would be as complex as figure 5.8.

While finite-state machines are relatively easy to implement and work well for simple tasks, they quickly become an impediment to even moderately complex tasks. Even for 2AFC tasks, many desirable features are difficult to implement with a finite state machine, such as: (1) graduation to a more difficult task depending on performance history, (2) adjusting reward volume based on learning rate, (3) selecting or synthesizing upcoming stimuli based on patterns of errors[221], etc.

Some of these problems are avoidable by using extended versions of finite state machines that allow for extra-state logic, but require additional complexity in the code running the state machines to accomodate, and with enough exceptions the clean systematicity that is the primary benefit of finite state machines is lost. Autopilot attempts to avoid these problems by providing *tools* to program tasks and describe them without *requiring a specified format*, balancing the increased complexity by scaffolding the broader ecosystem of the experiment like its output data, hardware control, etc. When possible, we have tried to avoid forcing people to change the way they think about their work to fit our "little universe"[4] and instead try to provide a set of tools that let researchers decide how they want to use them.

[4] We take inspiration from Aaron Swartz' description of another engineering project, the Semantic Web, that became too precious about its formalisms:

"Instead of the "let's just build something that works" attitude that made the Web (and the Internet) such a roaring success [...] they formed committees to form working groups to write drafts of ontologies that carefully listed (in 100-page Word documents) all possible things in the universe and the various properties they could have, and they spent hours in Talmudic debates over whether a washing machine was a kitchen appliance or a household cleaning device. [...] And instead of spending time building things, they've convinced people interested in these ideas that the first thing we need to do is write *standards.* (To engineers, this is absurd from the start—standards are things you write *after* you've got something working, not before!)"[222]

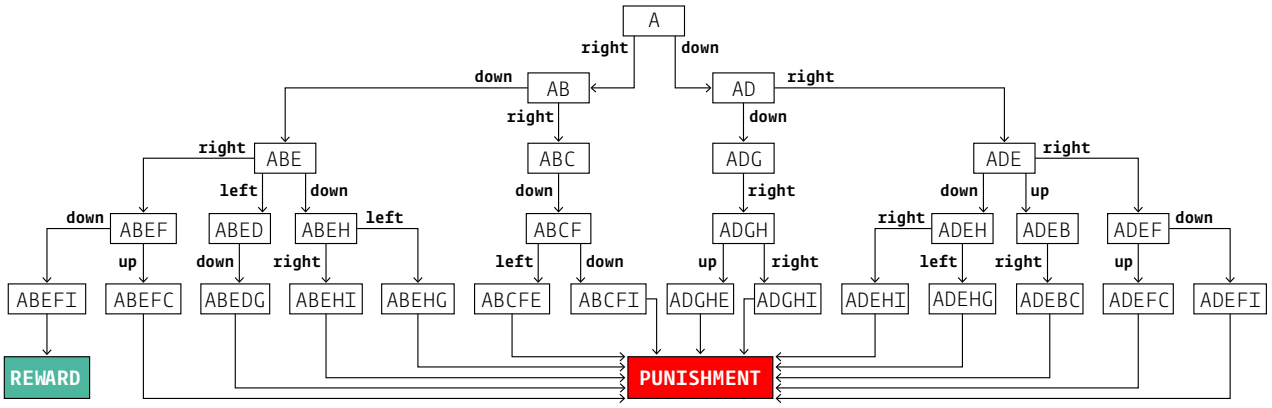


Figure 5.8: State transition tree for a simplified maze task.

5.4 *Hardware*

The Raspberry Pi can interface with nearly all common hardware, and has an extensive collection of guides, tutorials, and an active forum to support users implementing new hardware. There is also an enormous amount of existing hardware for the Raspberry Pi, including sound cards, motor controllers, sensor arrays, ADC/DACs, and touchscreen displays, largely eliminating the need for a separate ecosystem of purpose-built hardware (Table 5.1).

Autopilot controls hardware with an extensible inheritance hierarchy of Python classes intended to be built into a library of hardware controllers analogously to tasks. Autopilot uses pigpio to interact with its GPIO pins, giving Autopilot $5\mu s$ measurement precision and enabling protocols that require high precision (such as Serial, PWM, and I2C) for nearly all of the pins. Currently, Autopilot also has a family of objects to control cameras (both the Raspberry Pi Camera and high-speed GENICAM-compliant cameras), i2c-based motion and heat sensors, and USB mice. In the future we intend to improve performance further by replacing time-critical hardware operations with low-level interfaces written in Rust.

To organize and make available the vast amount of contextual knowledge needed to build and use experimental hardware, we have made a densely linked and publicly editable semantic wiki. The Autopilot wiki contains, among others, reference information for off-the-shelf parts, schematics for 2D and 3D-printable components, and guides for building experimental apparatuses and custom parts. The wiki combines unrestrictive freeform editing with structured, computer-readable semantic properties, and we have defined a collection of schemas for commonly documented items coupled with submission forms for ease of use. For example, the wiki page for the Lee Company solenoid we use has fields from a generic Part schema like a datasheet, price, and voltage, but also that it's a 3-way, normally-closed solenoid.

The wiki's blend of structure and freedom breaks apart typically monolithic hardware documentation into a collaborative, multimodal technical knowledge graph. Autopilot can access the wiki through its API, and we intend to tighten their integration over time, including automatic configurations for common parts, usage and longevity benchmarks, detecting mutually incompatible parts, and automatically resolving any additional plugins or dependencies needed to use a part.

Table 5.1: Cost of common peripherals. The native hardware of the Raspberry Pi, low-level hardware control of Autopilot, and availability of inexpensive off-the-shelf components compatible with the raspi make most custom-built peripherals unnecessary.

Device	Raspi	Bpod
HiFiBerry DAC2 Pro	$45	$445
ADC	$30	$495
I2C	$0	$225
Ethernet	$0	$285
Rotary Encoder	$0	$145

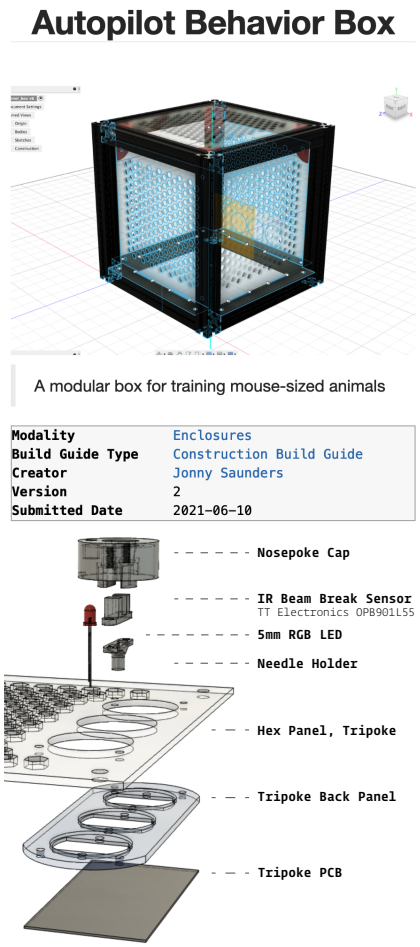


Figure 5.9: Two examples of parts with assembly guides available on the autopilot wiki: A modular behavior box with magnetic snap-in panels (top), and a three-nosepoke panel (bottom).

	Raspberry Pi 4B	Teensy 3.6	pyboard
CPU Clock	1.5GHz	180MHz	168MHz
CPU Cores	4	1	1
Architecture	ARMv8-A, 64-bit	ARMv7 32-bit	ARMv7 32-bit
RAM Size	2, 4, or 8GB	256KB	192KB
Storage	MicroSD (any size)	1024KB	1024KB
GPU	Broadcom VideoCore VI	—	—
GPIO Pins	40	58	29
USB Ports	2x USB 2.0, 2x USB 3.0	2x USB 2.0	1x USB 2.0
Ethernet	1Gbps	100Mbps	—
WiFi	2.4/5 GHz b/g/n/ac	—	—
Camera	15-pin Serial Interface	—	—
Bluetooth	✓	—	—

Table 5.2: Specifications of reviewed behavior hardware. BPod's state machine uses the Teensy 3.6 microcontroller, and PyControl uses the Micropython Pyboard.

5.5 *Transforms*

In v0.3.0, we introduced the `transform` module, a collection of tools for transforming data. The raw data off a sensor is often not in itself useful for performing an experiment: we want to compare it to some threshold, extract positions of objects in a video feed, and so on. Transforms are like building blocks, each performing some simple operation with a standard object structure, and then composed into a pipeline (Figure 5.10). Pipelines are portable, and can be created on the fly from a JSON representation of their arguments, so it's easy to offload expensive operations to a more capable machine for distributed realtime experimental control (See [205]).

In addition to computing derived values, we use transforms in a few ways, including

- **Bridging Hardware** — Different hardware devices use different data types, units, and scales, so transforms can rescale and convert values to make them compatible.
- **Integrating External Tools** — The number of exciting analytical tools for realtime experiments keep growing, but in practice they can be hard to use together. The transform module gives a scaffolding for writing wrappers around other tools and exposing them to each other in a shared framework, as we did with DeepLabCut-Live[205], making closed-loop pose tracking available to the rest of Autopilot's ecosystem. We don't need to rally thousands of independent developers to agree to write their tools in a shared library, instead transforms make wrapping them easy.
- **Extending Objects** — Transforms can be used to augment existing objects and create new ones. For example, a motion sensor uses the spheroid transform to calibrate its accelerometer, and the gammatone filter[5] extends the Noise sound to make a gammatone filtered noise sound.

Transform - DLC Live!

```
from autopilot import transform as t
# track points on a human body
dlc = t.image.DLC(
    model_zoo="full_human")
# select one of the knees
dlc += t.selection.DLCSlice(
    select="knee2")
# Test if it's in an ROI
dlc += t.logical.Condition(
    minimum=(0,0),
    maximum=(128,128))

# Process frames with the pipeline
# set pin High if knee2 in ROI
while True:
    pin.set(dlc.process(
        cam.q.get()))
```

Figure 5.10: Transforms can be chained together (here with the in-place addition operator `+=`) to make pipelines that encapsulate the logical relationship between some input and a desired output. Here `pin` is a Digital_Out object, and `cam` is a PiCamera with `queue` enabled.

[5] a thin wrapper around scipy's `signal.gammatone`

Like Tasks and Hardware, the transform module provides a scaffolding for writing reference implementations of algorithms commonly needed for realtime behavioral experiments. For example, neuroscientists often want to quickly measure a research subject's velocity or orientation, which is possible with inexpensive inertial motion sensors (IMUs), but since anything worth measuring will be swinging the sensor around with wild abandon the readings first need to be rotated back to a geocentric coordinate frame. Since the readings from an accelerometer are noisy, we found a few whitepapers describing using a Kalman filter for fusing the accelerometer and gyroscope data for a more accurate orientation estimate ([223, 224]), but couldn't find an implementation. We wrote one and integrated it into the IMU class (Figure 5.11). Since it's an independent transform, it's available to anyone even if they use nothing else from Autopilot.

Transforms were made to be composed, so we broke it into independent sub-operations: A Kalman filter, rotation, and a spheroid correction to calibrate accelerometers. Then we combined it with the DLC-Live transform for a fast but accurate motion estimate from position, velocity, and acceleration measurements from three independent sensors. Since each step of the transformation is exposed in a clean API, it was straightforward to extend the Kalman filter to accomodate the the wildly different sampling rates of the camera and IMU. It's still got its quirks, but that's the pur-

Geocentric Velocity

```
# rotate input in x and y
# by some pitch and roll
reorient = t.geometry.Rotate(
    dims='xy')
# select the z axis
reorient += t.selection.Slice(
    select=2)
# remove gravity
reorient += t.math.Add(
    -9.8)

# using I2C_9DOF ...
angle = imu.rotation
z_accel = reorient.process(
    imu._acceleration, angle)
```

Figure 5.11: Using the IMU_Orientation transform built into the IMU's `rotation` property, a processing chain to reorient the accelerometer reading and subtract gravity for geocentric z-axis acceleration.

pose of plugins — to make the code available and documented without formally integrating it in the library.

5.6 *Stimuli*

A hardware object would control a speaker, whereas stimulus objects are the individual sounds that the speaker would play. Like tasks and hardware, Autopilot makes stimulus generation portable between users, and is released with a family of common sounds like tones, noises, and sounds from files. The logic of sound presentation is contained in an inherited metaclass, so to program a new stimulus a user only needs to describe how to generate it from its parameters (Figure 5.12). Sound stimuli are better developed than visual stimuli as of v0.5.0, but we present a proof-of-concept visual experiment (Section 6.5) using psychopy[195].

An Autopilot Tone

```
my_tone = sounds.Tone(
    frequency = 500,
    duration  = 200)
my_tone.play()
```

Figure 5.12: Autopilot stimuli are parametrically defined and inherit all the playback logic that makes them easy to integrate in tasks

Autopilot controls the realtime audio server jack from an independent Python process that dumps samples directly into jack's buffer (Figure 5.13), giving it a trigger-to-playback latency very near the theoretical minimum (Section 6.2). Sounds can be pre-buffered in memory or synthesized on demand to play continuous sounds. Because the realtime server is independent from the logic of sound synthesis and storage, stimuli can be controlled independently from different threads without interrupting audio or dropping frames.

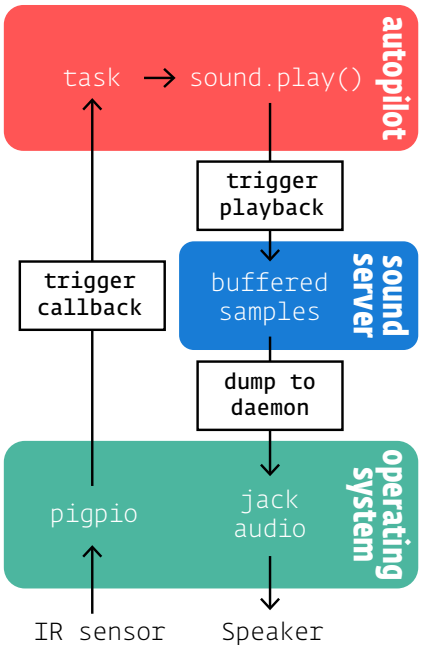


Figure 5.13: Our sound server keeps audio samples buffered until a `.play()` method is called, and then dumps them directly into the jack audio daemon.

We use the Hifiberry Amp 2, a combined sound card and amplifier, which is capable of 192kHz/24Bit audio playback. Autopilot and Jack can output to any sound hardware, however, including the builtin audio of the Raspberry Pi if fidelity isn't important. There are no external video cards for the Raspberry Pi 4b[6], but its embedded video card is capable of presenting video and visual stimuli (Section 6.5) especially if the other computationally demanding parts of the task are distributed to other Raspberry Pis (Section 5.7). If greater video performance is needed, Autopilot is capable of running on typical desktops as well as other single-board computers with GPUs (as we did with the Nvidia Jetson in [205]).

5.7 *Agents*

All of Autopilot's components can be organized into a single system as an "agent," the executable that coordinates everyday use. An agent encapsulates:

- **Runtime Logic** — an initialization routine that starts any needed system processes and any subsequent operations that define the behavior of the agent.
- **Networking Station** — Agents have networking objects called `Stations` that are intended to be the "load bearing" networking objects (described more below).
- **Callbacks** — An action vocabulary that maps different types of messages to methods for handling them. Called `listens` to disambiguate from other types of callbacks.
- **Dependencies** — Required packages, libraries, and system reconfigurations needed to operate. Python dependencies are currently defined for agents as groups of optional packages[7], and system configuration is done with scripts which shorthand common operations like compiling OpenCV with optimizations for the raspi or enabling a soundcard.

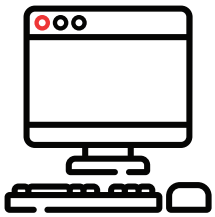
Together, these define an agent's *role* in the swarm.

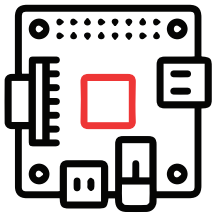
[6] though the Raspberry Pi compute module has a PCI lane that supports GPUs.

[7] As of v0.5.0, Autopilot is packaged with Poetry, so they are `[tool.poetry.extras]` entries within the `pyproject.toml` file, installed with pip like `pip install auto-pi-lot[pilot]` or poetry like `poetry install -E pilot`

There are currently two agents in Autopilot:

- **Terminal** - The user-facing control agent.
- **Pilot** - A Raspberry Pi that runs tasks, coordinates hardware, and optionally coordinates a set of child Pis.

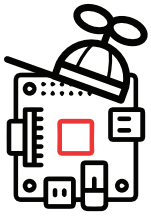




Terminal agents serve as a root node (see Section 5.8) in an Autopilot swarm. The terminal is the only agent with a GUI, which is used to control its connected pilots and visualize incoming task data. The terminal also manages data and keeps a registry of all active experimental subjects. The terminal is intended to make the day-to-day use of an Autopilot swarm manageable, even for those without programming experience. The terminal GUI is described further in Section 5.9.

Pilot agents are the workhorses of Autopilot—the agents that run the experiments. Pilots are intended to operate as always-on, continuously running system services. Pilots make a network connection to a terminal and wait for further instructions. They maintain the system-level software used for interfacing with the hardware connected to the Raspberry Pi, receive and execute tasks, and continually return data to the terminal for storage.

Each agent runs autonomously, and so a Pilot can run a task without a Terminal and store data locally, a Terminal can be used without Pilots to define protocols and manage subjects, and so on. This decoupling lets each agent have more freedom in its behavior at the expense of the complexity of configuring and maintaining them (see Sec. 7.10 and 7.13). All interaction is based on the "listen" callbacks known by the agents, so to start a task a Terminal will send a Pilot a `"START"` message containing information about a `Task` class that it is to run along with its parameterization. The Pilot then attempts to run the task, sends a message to the Terminal alerting it to a `"STATE"` change, and begins streaming data back to it in messages with a `"DATA"` key.



Each pilot is capable of mutually coordinating with one or many **Copilots**[8]. We are still experimenting with, and thus openminded to the best way to structure multi-pilot tasks. Like many things in Autopilot, there is no one right way to do it, and the strategy depends on the particular constraints of the task. We include a few examples in the network latency and go/no-go tasks in the plugin that accompanies this paper, and expand on this a bit further in a few parts of section 7, as it is a major point of active development.

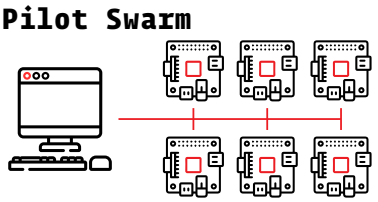
Behavioral topologies

We think one of the most transformative features of Autopilot's distributed structure is the control that users have over what we call "behavioral topology." The logic of hardware and task operation within an agent, the distribution of labor between agents performing a task, and the pattern of connectivity and command within a swarm of agents constitute a topology.

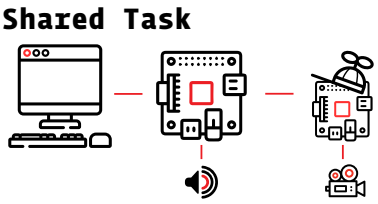
Below we illustrate this idea with a few examples:

[8] A previous version of this paper described a third, subordinate "Child" agent that performed auxiliary operations in a task. We now view such a hierarchy as unnecessary, and that distribution of labor within a task is better served by a fluid combination of multiple Pilots than thinking of them as qualitatively different agents. We now refer one among multiple agents performing a task together as a "copilot."

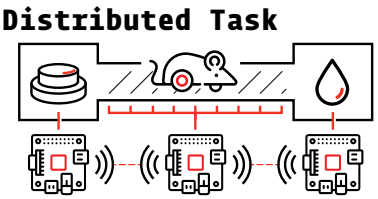
- **Pilot Swarm** - The first and most obvious topological departure from traditional behavioral instrumentation is the use of a single computer to independently coordinate tasks in parallel. Our primary installation of Autopilot is a cluster of 10 behavior boxes that can independently run tasks dispatched from a central terminal which manages data and visualization. This topology highlights the expandability of an Autopilot system: adding new pilots is inexpensive, and the single central terminal makes controlling experiments and managing data simple.



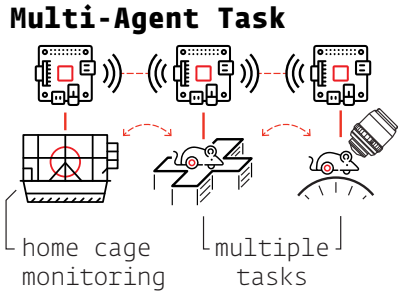
- **Shared Task** - Tasks can be shared across a set of copilots to handle tasks with computationally intensive operations. For example, in an open-field navigation task, one pilot can deliver position-dependent sounds while another records and analyzes video of the arena to track the animal's position. The terminal only needs to be configured to connect to the parent pilot, but the other copilot is free to send data to the Terminal marked for storage in the subject's file as well.



- **Distributed Task** - Many pilots with overlapping responsibilities can cooperate to perform distributed tasks. We anticipate this will be useful when the experimental arenas can't be fully contained (such as natural environments), or when experiments require simultaneous input and output from multiple subjects. Distributed tasks can take advantage of the Pi's wireless communication, enabling, for example, experiments that require many networked cameras to observe an area, or experiments that use the Pis themselves as an interface in a multisubject augmented reality experiment.



- **Multi-Agent Task** - Neuroscientific research often consists of multiple mutually interdependent experiments, each with radically different instrumentation. Autopilot provides a framework to unify these experiments by allowing users to rewrite core functionality of the program while maintaining integration between its components. For example, a neuroethologist could build a new "Observer" agent that continually monitors an animal's natural behavior in its home cage to calibrate a parameter in a task run by a pilot. If they wanted to manipulate the behavior, they could build a "Compute" agent that processes Calcium imaging data taken while the animal performs the task to generate and administer patterns of optogenetic stimulation. Accordingly, passively observed data can be combined with multiple experimental datasets from across the subject's lifespan. We think that unifying diverse experimental data streams with interoperable frameworks is the best way to perform experiments that measure natural behavior in the fullness of its complexity in order to understand the naturally behaving brain[200].



5.8 *Networking*

Agents use two types of object to communicate with one another: core **station** objects and peripheral **node** objects (Figure 5.14). Each agent creates one station in a separate process that handles all communication *between* agents. Stations are capable of forwarding data and maintaining agent state so the agent process is not unnecessarily interrupted. Nodes are created by individual modules run within an agent—eg. tasks, plots, hardware—that allow them to send and receive messages within an agent, or make connections directly to other nodes on other agents after the station discovers their network addresses. Messages are TCP packets[9], so there

[9] Autopilot uses ZeroMQ[206] and tornado to send and process messages

is no distinction between sending messages within a computer, a local network, or over the internet[10].

Both types of networking objects are tailored to their hosts by a set of callback functions — **listens** — that define how to handle each type of message. Messages have a uniform key-value structure, where the key indicates the listen used to process the message and the value is the message payload. This system makes adding new network-enabled components trivial:

Listing 7: A new networked LED

```
class LED_RGB(Hardware):
    def __init__(self):
        # call self.color for a 'COLOR' message
        self.listens = {'COLOR': self.color}
        self.node = networking.Node(
            id      = 'BEST_LED',
            listens = self.listens)

    def color(msg):
        self.set_color(msg.value)

# elsewhere in the code, we change the color to red!
node.send(to='BEST_LED', key='COLOR', value=[255,0,0])
```

Messages are serialized[11] with JSON, and can handle arrays, including on-the-fly compression with blosc. Net Nodes can create additional sockets to stream data that is stashed in a queue, and can take advantage of message batching and compressing multiple arrays together when latency is less critical.

Network connectivity is currently treelike by default (Figure 5.15) — each independent networking object can have many children but at most one parent. This structure makes an implicit assumption about the anisotropy of information flow: 'higher' nodes don't need to send messages to the 'lowest' nodes, and the 'lowest' nodes send all their messages to one or a few 'higher' nodes. It enforces simplified delegation of responsibilities in both directions: a terminal shouldn't need to know about every hardware object connected to all of its connected pilots, it just sends messages to the pilots, who handle it from there. A far-downstream node shouldn't need to know exactly how to send its data back to the terminal, so it pushes it upstream until it reaches a node that does.

This treelike structure is useful for getting started quickly with the default full system configuration, but for experimenting with different configurations it is also possible to directly connect network nodes. The Station backbone is then a useful way of connecting objects across agents, as messages can be sent as multihop messages through a connected tree of stations[12] to make an initial connection without hard-coding an IP or Port. In the future we plan to simplify this further by directly implementing a peer to peer discovery model, see Section 7.6.

[10] Though automatically configuring the use of faster protocols like IPC for communication within an agent or different backends like redis or gstreamer for data streams that would benefit from them is part of our development goals

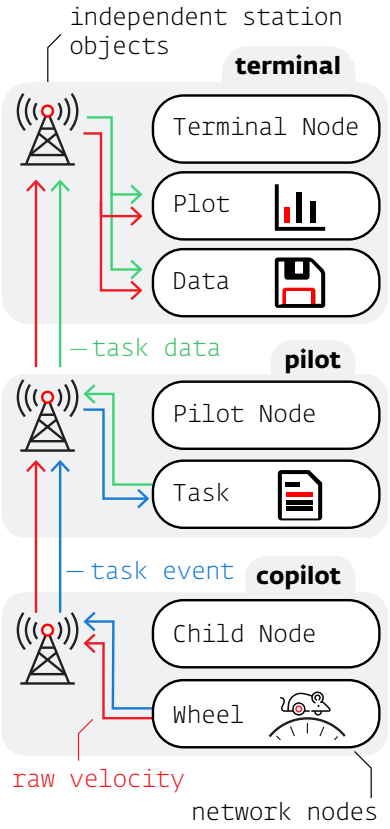


Figure 5.14: Autopilot segregates data streams efficiently—eg. raw velocity (red) can be plotted and saved by the terminal while only the task-relevant events (blue) are sent to the primary pilot. The pilot then sends trial-summarized data to the terminal (green).

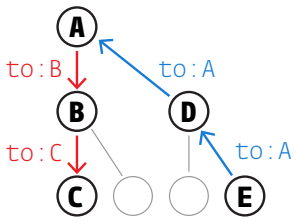


Figure 5.15: Treelike network structure—downstream messages are addressed by successive nodes, but upstream messages can always be pushed until the target is found.

[11] converted to binary suitable for sending between computers

[12] For example, to send a message from `E` to `C` in the diagram above:

```
node.send(to=["A", "B", "C"])
```

5.9 GUI & Plots

The terminal's GUI controls day-to-day system operation[13]. It is intended to be a nontechnical frontend that can be used by those without programming experience.

For each pilot, the terminal creates a control panel that manages subjects, task operation, and plots incoming data. Subjects can be managed through the GUI, including creation, protocol assignment, and metadata editing. Protocols can also be created from within the GUI. The GUI also has a set of basic maintenance and informational routines in its menus, like calibrating water ports or viewing a history of subject weights.

The simple callback design and network infrastructure makes adding new GUI functionality straightforward, and in the future we intend to extend the plugin system such that plugins can provide additional menu actions, plots, and utilities.

Plotting

Realtime data visualization is critical for monitoring training progress and ensuring that the task is working correctly, but each task has different requirements for visualization. A task that has a subject continuously running on a ball might require a continuous readout of running velocity, whereas a trial-based task only needs to show correct/incorrect responses as they happen. Autopilot approaches this problem by assigning the data returned by the task to graphical primitives like points, lines, or shaded areas as specified in a task's `PLOT` dictionary (taking inspiration from Wilkinson's grammar of graphics[225]).

The GUI is now some of the oldest code in the library, and we are in the process of decoupling some of its functionality from its visual representation and moving to a model where it is a thinner wrapper around the data modeling tools. Following the lead of formal models with strict typing will, for example, make plotting more fluid where the researcher can map incoming data to the set of graphical elements that are appropriate for its type. We discuss this further in section 7.8

[13] Autopilot uses PySide, a wrapper around Qt, to build its GUI.

Trial Plot

```
{"data": {
    "target"   : "point",
    "response" : "segment",
    "correct"  : "rollmean"
},
"roll_window" : 50}
```

Continuous Plot

```
{"data": {
    "target"   : "point",
    "response" : "segment",
    "velocity" : "shaded"
},
"continuous": true}
```

Figure 5.16: `PLOT` parameters for Figure 5.17. In both, "target" and "response" data are mapped to "point" and "segment" graphical primitives, but timestamps rather than trial numbers are used for the x-axis in the "continuous" plot (Figure 5.17, bottom). Additional parameters can be specified, eg. the trial plot (Figure 5.17, top) computes rolling accuracy over the past 50 trials

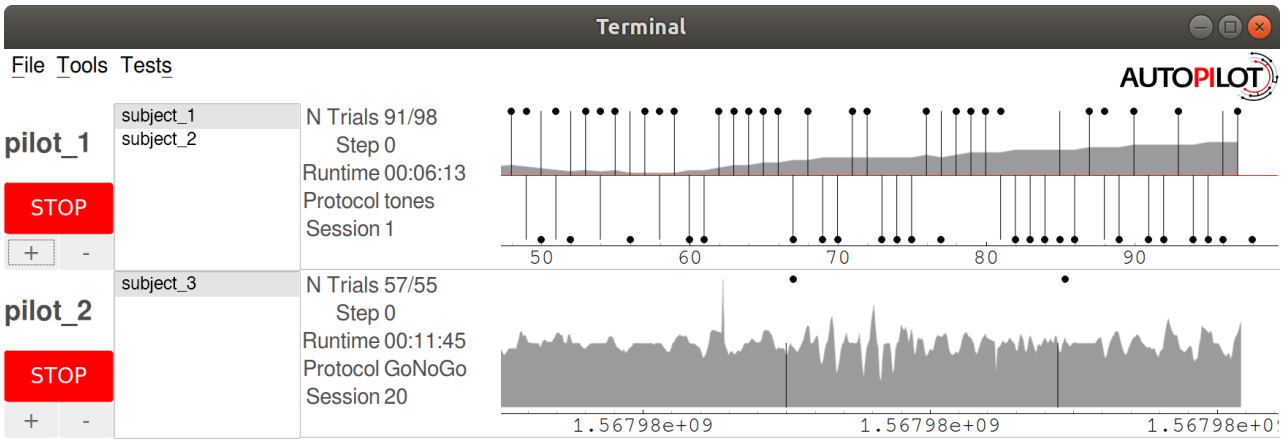


Figure 5.17: Screenshot from a terminal GUI running two different tasks with different plots concurrently. `pilot_1` runs 2 subjects: (`subject_1` and `subject_2`), while `pilot_2` runs `subject_3`. See Figure 5.16 for plot description

6

Tests

We have been testing and refining Autopilot since we built our swarm of 10 training boxes in the spring of 2019. In that time 178 mice[1] have performed over 6 million trials on a range of tasks. While Autopilot is still relatively new, it is by no means untested.

In this section we will present a set of basic performance benchmarks while also showing several of the different ways that Autopilot can be used. The code for all of the following tests is available as a plugin that is further documented on the wiki, and runs on a prerelease of v0.5.0. Materials tables (Table 6.1) for each test link more specifically to the test code and provide additional hardware and version documentation, where appropriate.

[1] All procedures were performed in accordance with National Institutes of Health guidelines, as approved by the University of Oregon Institutional Animal Care and Use Committee.

Table 6.1: General Materials

Hardware	
Raspi	Raspberry Pi 4b
Oscilloscope	Rigol DS1054Z
Software	
Autopilot	v0.5.0a
Plugin	Autopilot_Paper
Python	3.9.12
RaspiOS	Bullseye 22-04-04 (lite)
Analysis	
R	4.2.0
ggplot2[226]	3.3.5
dplyr[227]	1.0.9
purrr[228]	0.3.4
pandas[219]	1.4.2
numpy[198, 229]	1.21.6

6.1 GPIO Latency

Neurons compute at millisecond timescales, so any task that links neural computation to behavior needs to have near-millisecond latency. We start by characterizing Autopilot's GPIO control latency in "script mode" — using the GPIO control classes on their own, without using any of the rest of Autopilot's modules.

Output Latency

We first tested the software measured latency between when a command to write a value to a GPIO pin is issued and when it completes (Figure 6.1, Table 6.2). By default, the pigpio interface we use to control GPIO pins issues a command and then confirms the request was successful by querying the pigpio daemon for the status of the pin (Write+Read). We extended pigpio to just issue the command without confirmation to estimate the true time between when the command is issued and when the voltage of the pin changes (Write). Each of these operations takes roughly 40μs with minimal jitter (Median $\pm$ IQR — Write Only: 40.8$\mu s$$\pm$0.41, Write and Read: 88.4 $\mu s$$\pm$2.02, n=100,000 each).

Table 6.2: GPIO Latency Materials. (Parameters in {} are input in separate runs)

Code	test_gpio.py
replicate	`python test_gpio.py -w {0,1,2} -n 100000`
gpiozero	1.6.2
pigpio	3c23715

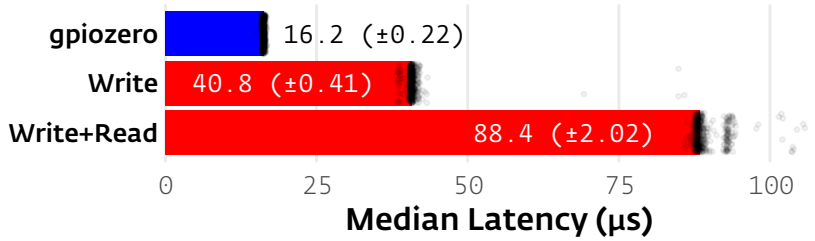


Figure 6.1: Software latency from GPIO write to completion of command. Values are presented as medians $\pm$ IQR with n=100,000 tests for each. A random subsample of 500 (for tractability of plotting) of each type of test are presented (black points) after filtering to the bottom 99th percentile to exclude extreme outliers. Commands sent using pigpio (red) took roughly 40μs each (write and read are effectively two separate commands, Write Only: 40.8$\mu s$$\pm$0.41, Write and Read: 88.4 $\mu s$$\pm$2.02). The prototype gpiozero wrapper (blue) using RPi.GPIO as its backend was faster, taking 16.2$\mu s$$\pm$0.22 to complete.

Pigpio is useful as a general purpose controller because of its ability to run scripts within its daemon, use hardware PWM via direct memory access, and consistently poll for pin state, but takes a latency penalty because the python interface communicates with it through a local TCP socket. To demonstrate the flexibility of Autopilot

in incorporating additional software libraries, we wrote a thin wrapper around gpiozero, which can use RPi.GPIO to directly write to the GPIO registers. For simple output, this wrapper proved to be faster (16.2 μs ± 0.22, n=100,000), and with 63 lines of code is now available in the plugin accompanying this paper to be used, repurposed, and extended.

Roundtrip Input/Output Latency

Output commands usually aren't issued in isolation, but as a response to some external or task-driven trigger. We measured the roundtrip latency from a 5V square pulse from an external function generator to when an output pin was flipped from low to high on an oscilloscope (Table 6.3).

Typically that is as much methodological detail as you would expect in a scientific paper, but actually making those measurements via oscilloscope requires knowing how to set up such a test as well as how to extract the measured data afterwards — which is not altogether trivially available technical knowledge. As an example of how integrating semantically linked documentation with experimental tools enables a fundamentally deeper kind of reproducibility and methodological transparency, we instead documented these operations, including a code sample and a guide to unlocking additional features on our oscilloscope on the autopilot wiki. The code to extract traces from the oscilloscope is also included in this paper's plugin, which links to the oscilloscope page with a `[[Controls Hardware::Rigol DS1054Z]]` tag, so it is possible to bidirectionally find code examples from the oscilloscope page as well as find further documentation about the hardware used in this paper from the plugin page. The same can be true for any hardware used by any plugin in any paper using Autopilot.

We used the `assign_cb` method of the `Digital_In` class to test the typical roundtrip latency that Autopilot objects can deliver. This gave us a median 474 μs (IQR: 52.5μs) latency (Red in Figure 6.3). GPIO callbacks are flexible, and can use arbitrary python functions, but if all that's needed is to trigger one pin off of another with some simple logic like a parametric digital waveform or static "on" time, pigpio also allows us to directly program pin to pin logic as a "pigs" script (literally Figure 6.2) that runs within the pigpio daemon. The pigs script gave us roughly three orders of magnitude lower latency (Median ± IQR: 370ns ± 140, blue in 6.3).

Table 6.3: Roundtrip Latency Materials

Function Generator	Koolertron CJDS98
Code	test_gpio.py
Replicate	`python test_gpio.py -w {3,4}`

Pigs Trigger Script

```
" ".join([
  "tag 999",
  # read input pin
  f"r {pin_in.pin_bcm}",
  # if off, goto 998
  f"jz 998",
  # else, turn on
  f"w {pin_out.pin_bcm} 1",
  # then goto 999
  "jp 999",
  "tag 998",
  # turn off
  f"w {pin_out.pin_bcm} 0",
  # jump to beginning
  f"jp 999"
])
```

Figure 6.2: The pigs script used to trigger one pin (pin_out), from another (pin_in). At the expense of a little bit of complexity having to write a script in its scripting language, we are able to reduce latency by three orders of magnitude.

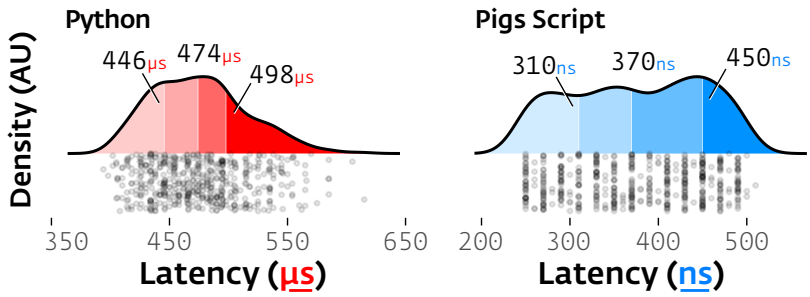


Figure 6.3: Roundtrip latency from external trigger to digital output using two methods: Typical Autopilot callback function given to a `Digital_In` object that turns a `Digital_Out` pin on for 1ms when an input pin changes state (Red, Left, Median ± IQR: 474 μs± 52.5). Pigs script that runs entirely within the pigpio daemon (Blue, Right, 380ns ± 140). For each, black points represent individual measurements (n=525), annotations are quartiles.

6.2 *Sound Latency*

We measured end to end, hardware input to sound output latency by measuring the delay between an external digital input and the onset of a 10kHz pure tone (Table 6.4). Sound playback was again triggered by the `Digital_In` class's callback method, and sound samples were buffered in a deque held in a separate process by the jack audio client between each trial. A `Digital_Out` pin was wired to the `Digital_In` pin in order to deliver the trigger pulse (but the `Digital_Out` pin was uninvolved in the software trigger for sound output).

Autopilot's jack audio backend was configured with a `192kHz` sampling rate with a buffer with two periods of of `32` samples each for theoretical minimum latency of `0.33ms`[2]. We observed a median 1.35ms (± 0.72 IQR) latency across 521 samples — roughly 4x the theoretical minimum (Figure 6.4). This suggests that Autopilot eliminates most perceptible end-to-end latency, which is necessary for tasks that require realtime feedback. One clear future direction is to write the sound processing loop in a compiled language exposed with a foreign function interface (FFI) to decrease both latency and jitter.

Table 6.4: Sound Latency Materials

Sound Card	Hifiberry Amp2
jack	1.9.22
Code	test_sound.py
Replicate	`python test_sound.py`

[2] A previous version of this paper included benchmarking and comparison to Bpod and pyControl's sound onset latency, but since then both packages have changed substantially, including Bpod creating a new hifi sound module based off HiFiBerry hardware very similar to the card used here, making those benchmarks obsolete. In this version we have omitted comparative benchmarks in favor of allowing the maintainers of those packages to publish their own benchmarks.

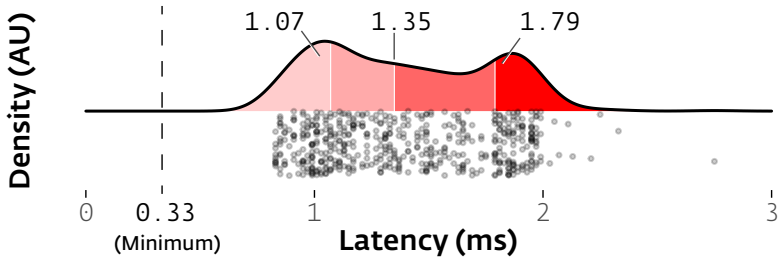


Figure 6.4: Autopilot has a median 1.35ms (± .72 IQR) latency between an external trigger and sound onset. Individual trials (dots, n=521) are shown beneath a density plot (red area under curve) colored by quartile (shades, numbers above are median, first, and third quartile). This latency is roughly 4x the theoretical minimum (0.33ms, dashed line).

6.3 *Network Latency*

To support data-intensive tasks like those that require online processing of video or electrophysiological data, the networking modules at the core of Autopilot need high bandwidth and low latency.

To test the latency of Autopilot's networking modules, we switch from "script mode" to "Task mode" (Table 6.5). Tasks are useful for encapsulating multistage routines across multiple devices that would be hard to coordinate with scripts alone. Our `Network_Latency` task consists of one "leader" pilot sending timestamped messages to a "follower" pilot which returns the timestamp marking when it received the message. The two pis communicate via two directly connected `Net_Nodes` (rather than routing each message through agent-level `Station` objects) after the leader pi initiates the follower with a multihop "START" message routed through a Terminal agent containing the task and networking parameters. We measured latency using software timestamps while synchronizing the clocks of the two pis with Chrony, an NTP daemon previously measured to synchronize Raspberry Pis within dozens of microseconds[230][3], with the leader pi hosting an NTP server and the follower pi synchronizing its clock solely from the leader. We documented this on the wiki too, since synchronization is a universal problem in multi-computer experiments.

Point to point latency was 0.975ms (median, ± 0.1 IQR, n=10,000, Figure 6.5) with some clear bimodality where a subset of messages (2,300 of 10,000) took longer

Table 6.5: Network Test Materials

Router	TP-Link AC1750
Chrony	4.0
zmq	22.3.0
Code	Network_Latency
Replicate	Assign task to subject from Terminal, start Task.

[3] Our sync is likely to be near to or better than that reported in [230]: in addition to a quiet network, we configured chrony to poll more frequently and tolerate a smaller error than default

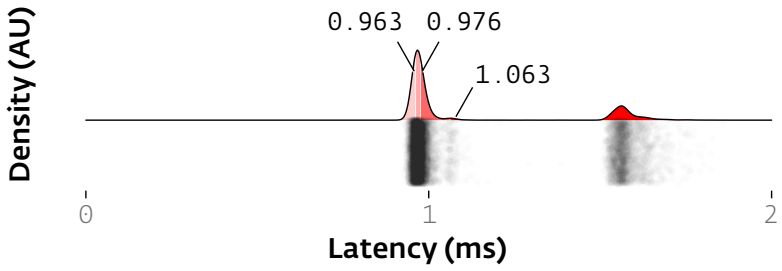


Figure 6.5: Network latency from when a message is sent from one pilot to when it is received by another. Messages took 0.975ms to send and receive (median, ± 0.1 IQR, n=10,000, overlaid numbers and red shading in density plot indicate quartiles). There is a clear bimodality in latencies for individual messages (black dots, jittered in y-axis) with unclear cause.

(median 1.567ms). The source of the bimodality is unclear to us, though it could be due to network congestion or interruption by other processes as the networking modules are not run in their own process like the sound server. This latency includes message serialization and deserialization by the builtin JSON library, which is on the order of roughly 100μs each for even the very small messages sent in this test. In future versions we will explore other serialization tools like msgpack and offer them as alternate serialization backends.

6.4 Network Bandwidth

To test Autopilot's bandwidth, we demonstrate yet another modality of use, using Autopilot's `Bandwidth_Test` widget, an action available from the Terminal GUI's `tests` menu that corresponds to a callback "listen" method in the Pilot (Table 6.6). This test requests that one or several pilots send messages at a range of selected frequencies and payload sizes back to the terminal. The messages pass through four networking objects en route: the stations and network nodes running the test for both the terminal and pilots (See Figure 5.14).

Table 6.6: Bandwidth Test Materials.

Terminal	Macbook Pro 2019, macOS 12.3.1, 2.4 GHz 8-Core Intel Core i9
blosc2	v0.2.0
Code	Bandwidth_Test, Pilot.l_bandwidth
Replicate	Terminal > Tools > Test Bandwidth

The needs for streaming experimental data vary depending on what is being streamed. Electrophysiological data is an n-electrode length vector sampled at a rate of dozens to hundreds of kilohertz, so each individual message isn't very large but there are a lot of them. Video data is a width by height (and for color video, by channel) array that can be relatively large[4], but it is captured at dozens to hundreds of hertz. Different data streams also have different degrees of compressibility: noisy, quasirandom electrical signals compress relatively poorly, while the typical behavioral neuroscientist's video of an animal that takes up 1/10th of the frame against a white background can have compression ratios in the hundreds.

Autopilot tries to provide flexibility for streaming different data types by offering message batching and optional on-the-fly compression with blosc. The bounds on bandwidth are then the speed at which an array can be compressed and the rate at which messages of a given size can be sent.

Autopilot's networking modules were able to send an "empty" (402 byte) message with headers describing the test but no payload at a maximum observed rate of 1,818Hz[5]. Approximately 15% of the duration is spent in message serialization, as a "frozen" preserialized message can be sent at 2,100Hz, though we imagine the need to send the same message thousands of times is rare.

We tested four types of messages with nonzero array[6] payloads: since the entropy of an array determines how compressible it is, we sent random and all-zero arrays with

[4] (1920 * 1080 * 3 * 8 bits) / 8 = 6 megabytes per frame of a 1080p color video, which is why video is rarely streamed uncompressed

[5] maximum average rate of 5000 messages for each of the equivalent empty message tests in the four conditions described below

[6] In all cases, float64 numpy arrays encoded in base64

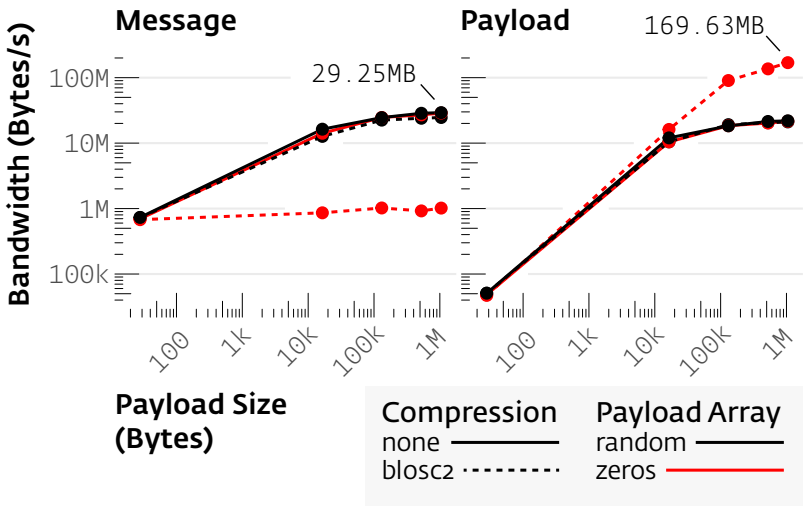


Figure 6.6: Bandwidth measurements between a pilot and terminal for compressed (solid lines) or uncompressed (dotted lines) arrays of random numbers (black) or zeros (red, each point n=5000 messages). As message size increased, the bandwidth for the rate of bytes transferred in serialized messages ("message bandwidth," left) plateaued at 29.25MBytes/s, while the effective bandwidth of arrays before and after compression ("payload bandwidth," right) reached 169.63MBytes/s. Real data will fall somewhere in this effective bandwidth range, depending on its compressibility.

and without compression. The random and all-zero arrays are the floor and ceiling of compressibility, respectively. Compression gives us two notions of bandwidth: the literal number of bytes that can be passed through a connection, and the effective bandwidth of the size of the arrays that can be transferred with a given compression ratio. We refer to these as "message" and "payload" bandwidth, respectively in Figure 6.6. Message bandwidth reflects the hardware limitations of the Raspberry Pi, but payload bandwidth is the number that matters in practice, as it measures the actual "speed of data" that can be used by the receiver.

As we increased the size of the array payload[7], the message bandwidth plateaued at a maximum of 29.25MByte per second (Figure 6.6, left). After this plateau, increasing the message size trades off linearly with the rate of messages sent. For all but the compressed array of zeros, the payload bandwidth mirrored the message bandwith with some trivial overhead from the base64 encoding. The compressed array of zeros, however, had an effective payload bandwidth of 169.6MBytes/s, a compromise between the speed of compression with the smaller message size[8]. The compressed random array had only negligible differences in payload and message bandwidth compared to the uncompressed random array, indicating that the overhead for blosc is trivial.

The ability to batch messages allows researchers to tune the size of an individual message to their particular need for high bandwidth or low latency. Since the compressibility of real data varies across the entire entropic range from randomness to arrays of all zeros, Autopilot doesn't have a single "bandwidth", but one that ranges between 30 and 170MByte/s[9].This bandwidth makes Autopilot capable of streaming raw Calcium imaging[10] and electrophysiological data from modern high-density probes[11]. Its flexible architecture allows researchers to decide how to build their experiments by distributing different components over different combinations of computers: stream data from a raspberry pi to a more powerful computer for processing, use GPIO rather than network triggers for time-critical operations — meeting the tooling challenge of complex, hardware-intensive, multimodal experiments that define contemporary systems neuroscience.

[7] n=5,000 for each condition at each size

[8] A message with a 1MByte zero array payload compressed to 6KBytes.

[9] In this dataset. There is additional payload bandwidth headroom with larger messages, and we include an additional dataset with a 200MByte/s bandwidth in the supplement.

[10] 2-Photon: 5.9MB/s
(12 bits * 512x512 resolution * 15Hz)

[11] Neuropixels: 14.4MB/s[181]
(10 bits * 30kHz * 384 channels)

6.5 *Distributed Go/No-go Task*

We designed a visual go/no-go task as a proof of concept for distributing task elements across multiple Pis, and also for the presentation of visual stimuli (Figure 6.7, Table 6.7). While the rest of the tests presented have been re-run, in the time since the intitial publication of the preprint we have not done substantial work on Autopilot's visual stimulus module, and so this section is presented as previously written using the v0.1.0 initial release.

In this task, a head-fixed subject would[12] be running on a wheel in front of a display with a lick-detecting water port able to deliver reward. Above the port is an LED. Whenever the LED is green, if the subject drops below a threshold velocity for a fixation period, a grating stimulus at a random orientation is presented on the monitor. After a random delay, there is a chance that the grating changes orientation by a random amount. If the subject licks the port in trials when the orientation is changed, or refrains from licking when it is not, the subject is rewarded.

One pilot controlled the operation of the task, including the coordination of a copilot. The pilot was connected to the LED and solenoid valve for reward delivery, as well as a monitor[13] to display the gratings[14]. The copilot continuously streamed velocity data (measured with a USB optical mouse against the surface of the wheel) back to the terminal for storage (see also Figure 5.14, which depicts the network topology for this task). The copilot waited for a message from the pilot to initiate measuring velocity, and when a rolling average of recent velocities fell below a given threshold the copilot sent a TTL trigger back to the pilot to start displaying the grating. This split-pilot topology allows us to poll the subject velocity continuously (at `125Hz` in this example) without competing for resources with psychopy's rendering engine.

We measured trigger (TTL pulse from the copilot) to visual stimulus onset latency using the measurement cursors of our oscilloscope as before. To detect the onset of the visual stimulus, we used a high-speed optical power meter attached to the top-left corner of our display monitor. The stimulus was a drifting Gabor grating drawn to fill half the horizontal and vertical width of the screen (`960 x 540px`), with a spatial frequency of `4cyc/960px` and temporal (drift) frequency of `1Hz`.

We observed a bimodal distribution of latencies (Quartiles: `28, 30, 36ms, n=50`, Figure 6.8), presumably because onsets of visual stimuli are quantized to the refresh rate (`60Hz, 16.67ms`) of the monitor. This range of latencies corresponds to the second and third frame after the trigger is sent (2/3 of observations fall in the 2nd frame, 1/3 of observations in the 3rd frame). We observed a median framerate of `36.2 FPS (IQR: 0.7)` across 50 trials (`8863` frames, Figure 6.9).

We further tested the Pi's framerate by using Psychopy's `timeByFrames` test—a script that draws stimuli without any Autopilot components running—to see if the framerate limits were imposed by the hardware of the Raspberry Pi or overhead from Autopilot (Table 6.8). We tested a series of Gabor filters and random dot stimuli (dots travel in random directions with equal velocity, default parameters) at different screen resolutions and stimulus complexities. The Raspberry Pi was capable of moderately high framerates (`>60 FPS`) for smaller, lower resolution stimuli, but struggled (`<30 FPS`) for full HD, fullscreen stimuli.

Table 6.7: Go/No-go Materials

Hardware	
Beam Break	TT Electronics OPB901L55
Monitor	Acer S230HL
Lick Port	Autopilot Tripoke v1
Light Sensor	Thorlabs PM100D
Software	
psychopy	v3.1.5
glfw	v1.8.3

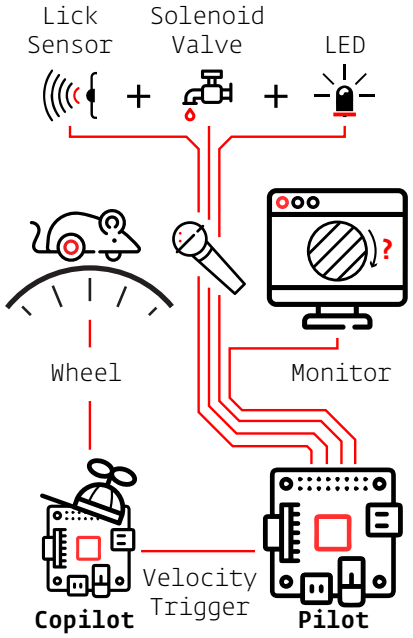


Figure 6.7: Hardware distribution for the distributed go/no-go task. Red lines indicate physical connections between hardware components. The lick sensor, solenoid valve, and LED are physically bundled into one component represented as the mouse's microphone.

[12] No mice were trained on this task

[13] (`1920×1080px, 60Hz`)

[14] Visual stimuli were presented with Psychopy using the glfw backend while Autopilot was run in a dedicated X11 server.

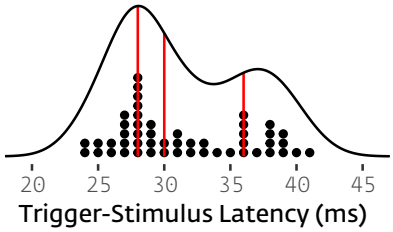


Figure 6.8: Stacked dots are a histogram of individual observations (n=50) underneath the probability density (black line), red lines indicate quartiles.

Autopilot is appropriate for realtime rendering of simple stimuli, and the proof-of-concept API we built around Psychopy doesn't impose discernible overhead (Mean framerate for a `960 x 540px` grating at `1080p` in Autopilot: `36.2 fps`, vs. `timeByFrames`: `35.0 fps`). In the future we will investigate prerendering and caching complex stimuli in order to increase performance. A straightforward option for higher-performance video would be to deploy an Autopilot agent running on a desktop computer with a high-performance GPU, or to use a single-board computer with a GPU like the NVIDIA Jetson ($99)[15].

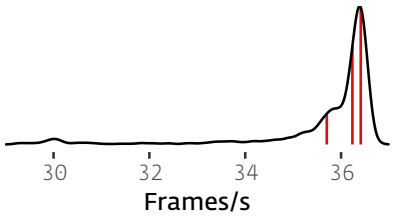


Figure 6.9: Probability density of framerates for 960 x 540px grating rendered at 1080p. Red lines indicate quartiles

[15] as we did in [205]

Stimulus	Resolution	Size / # Dots	Mean FPS	σ FPS
Gabor Filter	1280 x 720	300 x 300px	106.4	5.5
Gabor Filter	1920 x 1080	300 x 300px	75.2	3.5
Gabor Filter	1280 x 720	640 x 360px	53.5	2.2
Gabor Filter	1920 x 1080	960 x 540px	35.0	1.0
Gabor Filter	1280 x 720	720 x 720px	41.5	2.2
Gabor Filter	1920 x 1080	1080 x 1080px	20.1	0.7
Random Dots	1280 x 720	100 dots	98.0	3.8
Random Dots	1920 x 1080	100 dots	67.6	3.0
Random Dots	1280 x 720	1000 dots	20.9	0.25
Random Dots	1920 x 1080	1000 dots	19.5	0.36

Table 6.8: Tests performed over 1000 frames with PsychoPy's `timeByFrames` test.

7

Limitations and Future Directions

We will likely never view Autopilot as "finished." Autopilot—like all open-source software—is an evolving project, and this paper captures it as a snapshot at v0.5.0. We are invested in its development, and will be continually working to fix bugs, make its use more elegant, and add new features in collaboration with other researchers.

We expect that as the codebase matures and other researchers use Autopilot in new, unexpected ways that some fundamental elements of its structure may evolve. We have built version logging into the structure of the system so that changes will not compromise the replicability of experiments (see 7.5 below). While there will inevitably be breaking changes, these will be transparently documented, announced in release notes, and indicated with semantic versioning in order to alert users and describe how to adapt as needed.

We recognize the risk and inertia of retooling lab infrastructure, and there is still much work to be done on Autopilot. We welcome all issues and questions from anyone interested in contributing, or just curious to try it out — trying Autopilot is ultimately as risky as buying a Raspberry Pi.

The current major planned changes (also see the todo page in the docs) include:

1. **Python, Meet Rust** - Python is very useful as a high-level glue language, and its accessibility to a large number of scientific programmers is important to us, but it has its own very real performance limitations. As Autopilot's modules mature and stabilize, we are interested in rewriting core routines like sound presentation and networking in rust and exposing them to python with tools like PyO3
2. **Real Realtime** - Beneath user space decisions like programming language, the timing of CPU operations in linux is still determined by the kernel — this is one of the major reasons why other projects are based around dedicated microcontrollers. For almost everything that most scientists want to do, the standard linux kernel is perfectly fine, but we are interested in investigating what it would take to provide true deterministic realtime performance via Autopilot's high-level object system. One approach might be to provide prebuilt realtime kernel images along with tools to easily deploy them, though no firm plan has been made.
3. **Integration** - We will continue to collaborate with other programming teams to be interoperable with a broader array of other tools. Our next set of planned integrations include recording electrophysiological data by integrating with Open Ephys[231], optical imaging data from the Miniscope project [232, 233], and shared processing and control pipelines with Bonsai[193].
4. **Data Ingest & Export** - We are releasing Autopilot's data modeling system in v0.5.0 as an alpha release alongside this paper, and it includes prototype export interfaces to Neurodata Without Borders[207] and Datajoint[234]. Over the next several releases, we will continue to improve our data model so that researchers can easily structure their data and choose among different backends for storage. We are also working on a separate project to make tools to ingest data from the more ad-hoc directory-based data formats widely used in science and ingest them into Autopilot's and other tools formal modeling systems. In the longer term, we are interested in making Autopilot interoperable with linked data systems as part of a broader vision of digital infrastructure.
5. **Provenance** - Autopilot stores version information and local configuration in multiple places, and it is technically possible to faithfully replicate an experiment, but recording of provenance can still be consolidated and improved. By formalizing our object and data model, we will also systematize the many changes in configuration and version possible across the system for complete provenance tracking.
6. **P2P Networking** - The default tree structure of Autopilot's networking modules has proven to be unnecessarily limiting over time. In part, we had preoptimized for processing messages in a separate processes assuming that would help problems from dropped messages and overflowing send buffers, but in practice messages are almost never dropped and network nodes are as effective as stations in sending and receiving large amounts of data. As part of unifying Autopilot's object system, we will

implement a fully peer-to-peer networking system such that each instantiated object has a unique ID so that messages can be easily addressed from any object to any other in its swarm. We will learn from previous p2p addressing systems like distributed hash tables to allow net nodes to join the swarm and discover all other nodes automatically without manually configuring IP addresses and ports. In the longer term we are interested in peer to peer data transfer as well, so that an object serving as a data source can efficiently stream to many consumers without needing to duplicate each message for every consumer.

7. **Slots, Signals, and Streaming** - We will be supplementing a more general network structure with a system of specifying which attributes of each object are data sources, which are sinks, and what kind of connection they accept. Similar to Qt's signal and slot model, we want to make it as easy as using a `.connect()` method to control one piece of hardware with another. The transforms module should also be able to support branching and forking operations so multiple data sources can be combined for elaborated hardware control. ZeroMQ is an excellent tool for sending and receiving control messages, but formalized signals and slots could also specify different streaming tools like redis or gstreamer that might be better suited for high-bandwidth linear streams like video. Applied generally, this could also solve related problems like the relatively implicit handling of event triggers in the Task class and the need for manual configuration of connections between pilots and copilots.
8. **Rebuild the GUI** - The GUI is some of the oldest code in the library, was written before most of the other modules existed, and needs to be rebuilt. We have started by remaking its central widgets to be generated from pydantic models used increasingly throughout the system, but the rest of the GUI still needs to be rearchitected into a structure that decreases code duplication and allows us to do things like provide GUI extensions via plugins. We will likely continue to use Qt for the near future, but are also exploring the idea of webassembly tools to make browser-based web interfaces for remote control.
9. **Plugins** - We want Autopilot's plugin system to be permissive and as natural as the scripting style that most experimental code is written in, but we still need some means of specifying dependencies on other packages and plugins, among other improvements. We will be making a plugin generator that makes a folder of plugin boilerplate, as well as tools for installing, uploading, and synchronizing versions with git and the wiki. Over time we will make all object types within Autopilot able to be extended with plugins, as well as make it possible to override and extend built-in objects.
10. **Knowledge Organization** - We have been extending our thinking from code itself to more broadly consider the social systems that surround research code. The wiki was our first step, and we will continue to make more points of integration for smoothly incorporating contextual knowledge typically stored in lab notebooks into a public, collectively curated information system. We want to make it easier not only for individual researchers to use Autopilot, but make it easier for labs to coordinate work across projects without needing to rely on proprietary SaaS platforms with additional tooling for managing swarms, and moving beyond a single Autopilot wiki to a federated system of wikis for fluid continuity between "private" local coordination and "public" shared knowledge.
11. **Tests** - Our collection of tests doesn't cover the whole codebase, and so as we formalize our contribution process will move towards a system where all new code must have tests and documentation to be integrated. We also want to integrate our tests more closely with our documentation so that researchers know which part of the code has explicit tests guaranteeing functionality.
12. **Security** - Autopilot is a networked program, and while it doesn't execute arbitrary code from network messages, there is no security model to speak of. So far this hasn't beren a problem, as we encourage only using Autopilot on a local network behind a router, but as we we build out our networking modules we will investigate how to incorporate identity verification systems to protect swarms from malicious messages.
13. **Metastructure & API Maturity** - The scope and structure of Autopilot is still in flux relative to other, more mature Python packages. To reach a stable v1.0.0 API, we are in the process of unifying Autopilot's object structure so everything is clearly typed, all configuration is explicit, and all code written to handle special cases is absorbed into more general systems. Different parts of Autopilot have had different degrees of care over time, and so we will be working to catch the oldest modules up, trim unused ones, and make sure every line in the library is documented and useful. For the time being, flexibility is useful because frequently used or requested features trace a desire path outlining how its users believe Autopilot should behave. Each shortcoming we fix in Autopilot's modules makes it more straightforward to fix the rest, and so once the major remaining work is completed we will transition to a more conservative pace of development that ensures the longevity of the project.

8

Glossary

Agent	5.7	The executable part of Autopilot. A set of startup routines (eg. opening a GUI or starting an audio server), runtime behavior (eg. opening as a window or running as a background system process), and event handling methods (ie. **listens**) that constitute the role of the particular Autopilot instance in the **swarm**.
Copilot	5.7	An **agent** that performs some auxiliary, supporting role in a **task**—primarily used for offloading some hardware responsibilities from a **pilot**.
Graduation	5.3	Moving between successive **tasks** in a **protocol** when some criterion is met.
Listen	5.8	A method belonging to the **station** or **node** of a particular **agent** that defines how to process a particular type of message (ie. a message with a particular `key`).
Node	5.8	A networking object that some module (eg. hardware, **tasks**, GUI routines) or method (eg. a **listen**) uses to communicate with other **nodes**. Messages to other **agents** in the swarm are relayed through their **Station**
Pilot	5.7	An **agent** that runs on a Raspberry Pi, the primary experimental agent of Autopilot. Typically runs as a system service, receives **tasks** from a **terminal** and runs them. Can organize a group of **children** if requested by the **task**.
Protocol	5.3	A (`.json`) file that contains a list of **task** parameters and the **graduation** criteria to move between them. The **tasks** in a protocol are also known as its **levels**.
Stage	5.3	**Stages** are methods that implement the logic of a **task**. They can be used analogously to states in a finite-state machine (eg. wait for **trial** initiation, play stimulus, etc.) or asynchronously (whenever x input is received, rotate stimulus by y degrees).
Station	5.8	Each **agent** has a single **station**, a networking object that is run in its own process and is responsible for communication between **agents**. The **station** also routes messages from **children** or other **nodes**.
Swarm		Informally, a group of connected **agents**.
Task	5.3	A formalized description of an experiment: the parameters it takes, the data that it collects, the hardware it needs, and a collection of **stages** that describe what happens during the experiment.
Terminal	5.7	A user-facing **agent** that provides a GUI for operating and maintaining a **swarm**.
Topology	5.7	A particular combination of **agents**, their designated responsibilities, and the networking connections between them invoked by a **task** (eg. task requires one pilot to record video, one to process the video, and one to administer reward) or by usage (eg. 10 pilots are connected to a single terminal and are typically used to run 10 independent tasks, though they could run shared tasks together).
Trial	5.3	If a **task** is structured such that its **stages** form a repeating series, a **trial** is a single completion of that series.

Part III

Infrastructure

9

Bibliography

[1] Harvey M Sussman, David Fruchter, Jon Hilbert, and Joseph Sirosh. Linear correlates in the speech signal: The orderly output constraint. The Behavioral and brain sciences, 21(2):241–59; discussion 260–99, 1998. ISSN 0140-525X. https://doi.org/10.1017/S0140525X98001174. 1.1, 1.1, 1.2, 1.3

[2] L. L. Holt and A. J. Lotto. Speech perception as categorization. Attention, Perception & Psychophysics, 72(5):1218–1227, July 2010. ISSN 1943-3921. https://doi.org/10.3758/APP.72.5.1218. 1.1

[3] Yakov Kronrod, Emily Coppess, and Naomi H. Feldman. A unified account of categorical effects in phonetic perception. Psychonomic Bulletin & Review, 23(6):1681–1712, December 2016. ISSN 1069-9384. https://doi.org/10.3758/s13423-016-1049-y. 1.1, 2.2

[4] A M LIBERMAN, K S HARRIS, H S HOFFMAN, and B C GRIFFITH. The discrimination of speech sounds within and across phoneme boundaries. Journal of experimental psychology, 54(5):358–68, November 1957. ISSN 0022-1015. 1.1

[5] J L Elman and D Zipser. Learning the hidden structure of speech. The Journal of the Acoustical Society of America, 83 (4):1615–26, April 1988. ISSN 0001-4966. 1.1

[6] K R Kluender, R L Diehl, and P R Killeen. Japanese quail can learn phonetic categories. Science (New York, N.Y.), 237 (4819):1195–1197, September 1987. ISSN 0036-8075. https://doi.org/10.1126/science.3629235. 1.1, 1.1, 1.3, 1.4

[7] A M Liberman, F S Cooper, D P Shankweiler, and M Studdert-Kennedy. Perception of the speech code. Psychological review, 74(6):431–61, November 1967. ISSN 0033-295X. 1.1, 1.1

[8] E. Farnetani. V-C-V Lingual Coarticulation and Its Spatiotemporal Domain. In Speech Production and Speech Modelling, pages 93–130. Springer Netherlands, Dordrecht, 1990. https://doi.org/10.1007/978-94-009-2037-8_5. 1.1

[9] Randy L. Diehl, Andrew J. Lotto, and Lori L. Holt. Speech Perception. Annual Review of Psychology, 55(1):149–179, February 2004. ISSN 0066-4308. https://doi.org/10.1146/annurev.psych.55.090902.142028. 1.1, 1.1

[10] Joseph S. Perkell, Dennis H. Klatt, Kenneth N. Stevens, and Symposium on Invariance and Variability of Speech Processes (1983 : Massachusetts Institute of Technology). Invariance and Variability in Speech Processes. Lawrence Erlbaum Associates, 1986. ISBN 0-89859-545-2. 1.1

[11] Philip. Lieberman. The Biology and Evolution of Language. Harvard University Press, 1984. ISBN 0-674-07413-0. 1.1

[12] Alvin M. Liberman and Ignatius G. Mattingly. The motor theory of speech perception revised. Cognition, 21(1):1–36, October 1985. ISSN 00100277. https://doi.org/10.1016/0010-0277(85)90021-6. 1.1, 1.1

[13] Kathy M. Carbonell and Andrew J. Lotto. Speech is not special... again. Frontiers in psychology, 5(June):427, June 2014. ISSN 1664-1078. https://doi.org/10.3389/fpsyg.2014.00427. 1.1, 2.2

[14] Asif A. Ghazanfar and Marc D. Hauser. The neuroethology of primate vocal communication: Substrates for the evolution of speech. Trends in Cognitive Sciences, 3(10):377–384, 1999. ISSN 13646613. https://doi.org/10.1016/S1364-6613(99)01379-0. 1.1

[15] Ina Bornkessel-Schlesewsky, Matthias Schlesewsky, Steven L. Small, and Josef P. Rauschecker. Neurobiological roots of language in primate audition: Common computational properties. Trends in Cognitive Sciences, 19(3):142–150, March 2015. ISSN 13646613. https://doi.org/10.1016/j.tics.2014.12.008. 1.1

[16] K R Kluender and a J Lotto. Effects of first formant onset frequency on [-voice] judgments result from auditory processes not specific to humans. The Journal of the Acoustical Society of America, 95(2):1044–52, 1994. ISSN 0001-4966. https://doi.org/10.1121/1.408466. 1.1

[17] Patricia K. Kuhl and James D. Miller. Speech perception by the chinchilla: Identification functions for synthetic VOT stimuli. The Journal of the Acoustical Society of America, 63(3):905–917, 1978. ISSN 00014966. https://doi.org/10.1121/1.381770. 1.1, 1.3

[18] Crystal T. Engineer, Kimiya C. Rahebi, Elizabeth P. Buell, Melyssa K. Fink, and Michael P. Kilgard. Speech training alters consonant and vowel responses in multiple auditory cortex fields. Behavioural Brain Research, 287:256–264, 2015. ISSN 18727549. https://doi.org/10.1016/j.bbr.2015.03.044. 1.1, 1.2, 1.3, 1.4

[19] P K Kuhl and D M Padden. Enhanced discriminability at the phonetic boundaries for the place feature in macaques. The Journal of the Acoustical Society of America, 73(3):1003–1010, 1983. ISSN 0001-4966. https://doi.org/10.3758/BF03204208. 1.1, 1.3

[20] Robert J Dooling, Catherine T. Best, and Susan D. Brown. Discrimination of synthetic full-formant and sinewave /ra–la/ continua by budgerigars (Melopsittacus undulatus) and zebra finches (Taeniopygia guttata). The Journal of the Acoustical Society of America, 97(3):1839–1846, 1995. ISSN 00014966. https://doi.org/10.1121/1.412058. 1.1, 1.3

[21] Keith R. Kluender. Contributions of nonhuman animal models to understanding human speech perception. The Journal of the Acoustical Society of America, 107(5):2835–2835, May 2000. ISSN 0001-4966. https://doi.org/10.1121/1.429153. 1.1, 1.3, 2.2

[22] Jennifer K Bizley and Yale E Cohen. The what, where and how of auditory-object perception. Nature reviews. Neuroscience, 14(10):693–707, October 2013. ISSN 1471-0048. https://doi.org/10.1038/nrn3565. 1.1

[23] Josef P Rauschecker and Sophie K Scott. Maps and streams in the auditory cortex: Nonhuman primates illuminate human speech processing. Nature Neuroscience, 12(6):718–724, June 2009. ISSN 1097-6256. https://doi.org/10.1038/nn.2331. 1.1

[24] Ted J. Strauss, Harlan D. Harris, and James S. Magnuson. jTRACE: A reimplementation and extension of the TRACE model of speech perception and spoken word recognition. Behavior Research Methods, 39(1):19–30, February 2007. ISSN 1554-351X. https://doi.org/10.3758/BF03192840. 1.1

[25] Keith R Kluender, Christian E Stilp, Michael Kiefte, K R Kluender, C E Stilp, and M Kiefte. Perception of Vowel Sounds Within a Biologically Realistic Model of Efficient Coding. In Vowel Inherent Spectral Change, pages 117–151. 2013. ISBN 978-3-642-14209-3. https://doi.org/10.1007/978-3-642-14209-3_6. 1.1

[26] M. Gareth Gaskell and William D. Marslen-Wilson. Integrating Form and Meaning: A Distributed Model of Speech Perception. Language and Cognitive Processes, 12(5-6):613–656, October 1997. ISSN 0169-0965. https://doi.org/10.1080/016909697386646. 1.1

[27] D. H. Hubel and T. N. Wiesel. Receptive fields, binocular interaction and functional architecture in the cat's visual cortex. The Journal of Physiology, 160(1):106–154.2, January 1962. ISSN 0022-3751. https://doi.org/10.1523/JNEUROSCI.1991-09.2009. 1.1

[28] K. G. Ranasinghe, W. A. Vrana, C. J. Matney, and M. P. Kilgard. Increasing diversity of neural responses to speech sounds across the central auditory pathway. Neuroscience, 252:80–97, 2013. ISSN 03064522. https://doi.org/10.1016/j.neuroscience.2013.08.005. 1.1

[29] Edward L. Bartlett. The organization and physiology of the auditory thalamus and its role in processing acoustic features important for speech perception. Brain and language, 126(1):29–48, July 2013. ISSN 10902155. https://doi.org/10.1016/j.bandl.2013.03.003. 1.1

[30] T. M. Centanni, A. M. Sloan, A. C. Reed, C. T. Engineer, R. L. Rennaker, and M. P. Kilgard. Detection and identification of speech sounds using cortical activity patterns. Neuroscience, 258:292–306, 2013. ISSN 03064522. https://doi.org/10.1016/j.neuroscience.2013.11.030. 1.1

[31] Crystal T Engineer, Claudia A Perez, YeTing H Chen, Ryan S Carraway, Amanda C Reed, Jai A Shetake, Vikram Jakkamsetti, Kevin Q Chang, and Michael P Kilgard. Cortical activity patterns predict speech discrimination ability. Nature neuroscience, 11(5):603–8, May 2008. ISSN 1097-6256. https://doi.org/10.1038/nn.2109. 1.1

[32] Mitchell Steinschneider, Yonatan I Fishman, and Joseph C Arezzo. Representation of the voice onset time (VOT) speech parameter in population responses within primary auditory cortex of the awake monkey. The Journal of the Acoustical Society of America, 114(1):307–321, 2003. ISSN 00014966. https://doi.org/10.1121/1.1582449. 1.1

[33] Nima Mesgarani, Connie Cheung, Keith Johnson, and Edward F Chang. Phonetic feature encoding in human superior temporal gyrus. Science (New York, N.Y.), 343(6174):1006–10, 2014. ISSN 1095-9203. https://doi.org/10.1126/science.1245994. 1.1

[34] P Belin, R J Zatorre, P Lafaille, P Ahad, and B Pike. Voice-selective areas in human auditory cortex. Nature, 403(6767):309–312, 2000. ISSN 0028-0836. https://doi.org/10.1038/35002078. 1.1

[35] Edward F Chang, Jochem W Rieger, Keith Johnson, Mitchel S Berger, Nicholas M Barbaro, and Robert T Knight. Categorical speech representation in human superior temporal gyrus. Nature neuroscience, 13(11):1428–32, November 2010. ISSN 1546-1726. https://doi.org/10.1038/nn.2641. 1.1

[36] Brian N. Pasley, Stephen V. David, Nima Mesgarani, Adeen Flinker, Shihab A. Shamma, Nathan E. Crone, Robert T. Knight, and Edward F. Chang. Reconstructing speech from human auditory cortex. PLoS Biology, 10(1), 2012. ISSN 15449173. https://doi.org/10.1371/journal.pbio.1001251. 1.1

[37] Gavin M. Bidelman, Sylvain Moreno, and Claude Alain. Tracing the emergence of categorical speech perception in the human auditory system. NeuroImage, 79:201–212, 2013. ISSN 10538119. https://doi.org/10.1016/j.neuroimage.2013.04.093. 1.1

[38] Andrew Ng and Michael I. Jordan. On generative vs. discriminative classifiers: A comparison of logistic regression and naive bayes. Proceedings of Advances in Neural Information Processing, 28(3):169–187, 2002. ISSN 13704621. https://doi.org/10.1007/s11063-008-9088-7. 1.1

[39] Ferdinand de Saussure. Cours de Linguistique Générale. Payot, Lausanne, Paris, 1916. 1.1

[40] Ueli Rutishauser, Jean Jacques Slotine, and Rodney Douglas. Computation in Dynamically Bounded Asymmetric Systems. PLoS Computational Biology, 11(1):e1004039, 2015. ISSN 15537358. https://doi.org/10.1371/journal.pcbi.1004039. 1.1

[41] B Elan Dresher. The contrastive hierarchy in phonology. Contrast in phonology: theory, perception, acquisition, 13:11, 2008. ISSN 1718-3510. https://doi.org/10.1017/CBO9780511642005. 1.1, 2.2

[42] Helen Blank and Matthew H Davis. Prediction Errors but Not Sharpened Signals Simulate Multivoxel fMRI Patterns during Speech Perception. PLoS Biology, 14(11):e1002577, November 2016. ISSN 1545-7885. https://doi.org/10.1371/journal.pbio.1002577. 1.1

[43] Pierre Gagnepain, Richard N. Henson, and Matthew H. Davis. Temporal Predictive Codes for Spoken Words in Auditory Cortex. Technical Report 7, 2012. 1.1

[44] Neal P. Fox and Sheila E. Blumstein. Top-down effects of syntactic sentential context on phonetic processing. Journal of Experimental Psychology: Human Perception and Performance, 42(5):730–741, May 2016. ISSN 1939-1277. https://doi.org/10.1037/a0039965. 1.1

[45] Bert Schouten, Ellen Gerrits, and Arjan Van Hessen. The end of categorical perception as we know it. In Speech Communication, volume 41, pages 71–80, 2003. ISBN 0167-6393. https://doi.org/10.1016/S0167-6393(02)00094-8. 1.1

[46] Lawrence D. Rosenblum. Speech Perception as a Multimodal Phenomenon. Current Directions in Psychological Science, 17(6):405–409, December 2008. ISSN 0963-7214. https://doi.org/10.1111/j.1467-8721.2008.00615.x. 1.1

[47] PK Kuhl, KA Williams, F Lacerda, KN Stevens, and B Lindblom. Linguistic experience alters phonetic perception in infants by 6 months of age. Science, 255(5044), 1992. 1.1

[48] E. A. Brenowitz, D. Margoliash, and K. W. Nordeen. An introduction to birdsong and the avian song system. Journal of Neurobiology, 33(5):495–500, 1997. ISSN 00223034. 1.1

[49] Frédéric E Theunissen and Julie E Elie. Neural processing of natural sounds. Nature reviews. Neuroscience, 15(6): 355–66, 2014. ISSN 1471-0048. https://doi.org/10.1038/nrn3731. 1.1

[50] Gordon E Peterson and Harold L Barney. Control methods used in a study of the vowels. The Journal of the Acoustical Society of America, 24(2):175–184, 1952. ISSN 00014966. https://doi.org/10.1121/1.1906875. 1.1

[51] Björn Lindblom and Harvey M. Sussman. Dissecting coarticulation: How locus equations happen. Journal of Phonetics, 40(1):1–19, 2012. ISSN 00954470. https://doi.org/10.1016/j.wocn.2011.09.005. 1.2

[52] S. Sadagopan and X. Wang. Nonlinear Spectrotemporal Interactions Underlying Selectivity for Complex Sounds in Auditory Cortex. Journal of Neuroscience, 29(36):11192–11202, September 2009. ISSN 0270-6474. https://doi.org/10.1523/JNEUROSCI.1286-09.2009. 1.3

[53] Xiaoqin Wang, Thomas Lu, Ross K Snider, and Li Liang. Sustained firing in auditory cortex evoked by preferred stimuli. Nature, 435(7040):341–6, 2005. ISSN 1476-4687. https://doi.org/10.1038/nature03565. 1.3, 2.2

[54] Kelly E Radziwon, Kristie M June, Daniel J Stolzberg, Matthew A Xu-Friedman, Richard J Salvi, and Micheal L Dent. Behaviorally measured audiograms and gap detection thresholds in CBA/CaJ mice. Journal of comparative physiology. A, Neuroethology, sensory, neural, and behavioral physiology, 195(10):961–9, October 2009. ISSN 1432-1351. https://doi.org/10.1007/s00359-009-0472-1. 1.4

[55] Paul Boersma. Praat, a system for doing phonetics by computer. Glot International, 5(9/10):341–347, 2001. ISSN 0196-0202. https://doi.org/10.1097/AUD.0b013e31821473f7. 1.4

[56] H. D. Patterson and R. Thompson. Recovery of Inter-Block Information when Block Sizes are Unequal. Biometrika, 58 (3):545, December 1971. ISSN 00063444. https://doi.org/10.2307/2334389. 1.4

[57] R Core Team. R: A language and environment for statistical computing., 2016. 1.4

[58] RStudio Team. RStudio: Integrated Development for R., 2015. 1.4

[59] Douglas Bates, Martin Machler, Ben Bolker, and Steve Walker. Fitting Linear Mixed-Effects Models Using lme4. Journal of Statistical Software, 67(1):1–48, 2015. https://doi.org/10.18637/jss.v067.i01. 1.4

[60] Martin Maechler, Peter Rousseeuw, Anja Struyf, Mia Hubert, and Kurt Hornik. Cluster: Cluster Analysis Basics and Extensions, 2017. 1.4

[61] Dorai-Raj Sundar. Binom: Binomial Confidence Intervals For Several Parameterizations, 2014. 1.4

[62] Hadley Wickham. The Split-Apply-Combine Strategy for Data Analysis. Journal of Statistical Software, 40(1):1–29, 2011. 1.4

[63] Hadley Wickham. Ggplot2: Elegant Graphics for Data Analysis. Springer-Verlag, New York, NY, 2009. ISBN 978-0-387-98140-6. 1.4

[64] David B. Dahl. Xtable: Export Tables to LaTeX or HTML, 2016. 1.4

[65] Ludwig Wittgenstein. Philosophical Investigations. Basil Blackwell, Oxford, 1968. ISBN 978-0-631-11900-5. 2.2

[66] Leigh Lisker. Rapid versus rabid: A catalogue of acoustic features that may cue the distinction. The Journal of the Acoustical Society of America, 62(S1):S77, 1977. ISSN 00014966. https://doi.org/10.1121/1.2016377. 2.2, 2.2

[67] P J Bailey and Q Summerfield. Information in speech: Observations on the perception of [s]-stop clusters. Journal of experimental psychology. Human perception and performance, 6(3):536–563, August 1980. ISSN 0096-1523. https://doi.org/10.1037/0096-1523.6.3.536. 2.2, 2.2, 2.2, 2.5

[68] Eleanor Rosch and Carolyn B Mervis. Family resemblances: Studies in the internal structure of categories. Cognitive Psychology, 7(4):573–605, October 1975. ISSN 0010-0285. https://doi.org/10.1016/0010-0285(75)90024-9. 2.2, 2.2

[69] S Edelman. Representation is representation of similarities. The Behavioral and brain sciences, 21(4):449–67; discussion 467–98, August 1998. ISSN 0140-525X. 2.2

[70] Amos Tversky. Features of Similarity. 84(4), 1977. 2.2, 2.2

[71] Michael D. Lee and Daniel J. Navarro. Extending the ALCOVE model of category learning to featural stimulus domains. Psychonomic Bulletin & Review, 9(1):43–58, March 2002. ISSN 1531-5320. https://doi.org/10.3758/BF03196256. 2.2, 2.2

[72] Jessamyn Schertz and Emily J. Clare. Phonetic cue weighting in perception and production. WIREs Cognitive Science, 11(2):e1521, 2020. ISSN 1939-5086. https://doi.org/10.1002/wcs.1521. 2.2, 2.2

[73] John J Ohala, Arthur Bronstein, J, M. Grazia Busà, Julie A Lewis, and William F Weigel, editors. A Guide to the History of the Phonetic Sciences in the United States. 1999. 2.2

[74] Alvin M. Liberman and Ignatius G. Mattingly. The motor theory of speech perception revised. Cognition, 21(1):1–36, 1985. ISSN 00100277. https://doi.org/10.1016/0010-0277(85)90021-6. 2.2

[75] Haskins Laboratories. https://web.archive.org/web/20200809223413/http://www.haskins.yale.edu/featured/sws/sws.html, August 2020. 2.2

[76] Patricia K. Kuhl. A new view of language acquisition. Proceedings of the National Academy of Sciences, 97(22):11850–11857, October 2000. ISSN 0027-8424, 1091-6490. https://doi.org/10.1073/pnas.97.22.11850. 2.2

[77] Patricia K. Kuhl. Early language acquisition: Cracking the speech code. Nature Reviews Neuroscience, 5(11):831–843, November 2004. ISSN 1471-0048. https://doi.org/10.1038/nrn1533. 2.2, 2.2

[78] Robert E. Remez, Philip E. Rubin, Stefanie M. Berns, Jennifer S. Pardo, and Jessica M. Lang. On the perceptual organization of speech. Psychological Review, 101(1):129–156, January 1994. ISSN 0033-295X. https://doi.org/10.1037/0033-295X.101.1.129. 2.2

[79] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention Is All You Need. arXiv:1706.03762 [cs], December 2017. 2.2

[80] G.N. Clements. Feature Organization. In Encyclopedia of Language & Linguistics, pages 433–440. Elsevier, 2006. ISBN 978-0-08-044854-1. https://doi.org/10.1016/B0-08-044854-2/00055-9. 2.2

[81] Morris Halle, Bert Vaux, and Andrew Wolfe. On Feature Spreading and the Representation of Place of Articulation. Linguistic Inquiry, 31(3):387–444, July 2000. ISSN 0024-3892. https://doi.org/10.1162/002438900554398. 2.2

[82] Pavel Iosad. Vowel reduction in Russian: No phonetics in phonology. 2012. https://doi.org/10.1017/S0022226712000102. 2.1, 2.2

[83] Danielle J. Navarro. Between the Devil and the Deep Blue Sea: Tensions Between Scientific Judgement and Statistical Model Selection. Computational Brain & Behavior, 2(1):28–34, March 2019. ISSN 2522-087X. https://doi.org/10.1007/s42113-018-0019-z. 2.2

[84] Mona Lindau. The story of /r/. The Journal of the Acoustical Society of America, 67(S1):S27–S27, April 1980. ISSN 0001-4966. https://doi.org/10.1121/1.2018134. 2.2

[85] A. M. Liberman. Some characteristics of perception in the speech mode. Research Publications - Association for Research in Nervous and Mental Disease, 48:238–254, 1970. ISSN 0091-7443. 2.2

[86] Keith R. Kluender, Christian E. Stilp, and Michael Kiefte. Perception of Vowel Sounds Within a Biologically Realistic Model of Efficient Coding. In Geoffrey Stewart Morrison and Peter F. Assmann, editors, Vowel Inherent Spectral Change, pages 117–151. Springer Berlin Heidelberg, Berlin, Heidelberg, 2013. ISBN 978-3-642-14208-6 978-3-642-14209-3. https://doi.org/10.1007/978-3-642-14209-3_6. 2.2

[87] Christian E. Stilp and Keith R. Kluender. Efficient Coding and Statistically Optimal Weighting of Covariance among Acoustic Attributes in Novel Sounds. PLoS ONE, 7(1):e30845, January 2012. ISSN 1932-6203. https://doi.org/10.1371/journal.pone.0030845. 2.2, 2.2

[88] C. E. Stilp, T. T. Rogers, and K. R. Kluender. Rapid efficient coding of correlated complex acoustic properties. Proceedings of the National Academy of Sciences, 107(50):21914–21919, December 2010. ISSN 0027-8424, 1091-6490. https://doi.org/10.1073/pnas.1009020107. 2.2, 2.2

[89] Evan C. Smith and Michael S. Lewicki. Efficient auditory coding. Nature, 439(7079):978–982, February 2006. ISSN 1476-4687. https://doi.org/10.1038/nature04485. 2.2, 2.2, 2.3

[90] Maria N Geffen, Judit Gervain, Janet F Werker, and Marcelo O Magnasco. Auditory perception of self-similarity in water sounds. Front Integr Neurosci, 5(May):15, 2011. ISSN 1662-5145. https://doi.org/10.3389/fnint.2011.00015. 2.2

[91] Michael Kiefte and Keith R. Kluender. Absorption of reliable spectral characteristics in auditory perception. The Journal of the Acoustical Society of America, 123(1):366–376, January 2008. ISSN 0001-4966. https://doi.org/10.1121/1.2804951. 2.2

[92] Shi Tong Liu, Pilar Montes-Lourido, Xiaoqin Wang, and Srivatsun Sadagopan. Optimal features for auditory categorization. Nature Communications, 10(1):1302, March 2019. ISSN 2041-1723. https://doi.org/10.1038/s41467-019-09115-y. 2.2, 2.3

[93] Paul Iverson and Patricia K. Kuhl. Influences of phonetic identification and category goodness on American listeners' perception of /r/ and /l/. The Journal of the Acoustical Society of America, 99(2):1130–1140, February 1996. ISSN 0001-4966. https://doi.org/10.1121/1.415234. 2.2

[94] Pamela Souza, Richard Wright, Frederick Gallun, and Paul Reinhart. Reliability and Repeatability of the Speech Cue Profile. Journal of speech, language, and hearing research: JSLHR, 61(8):2126–2137, August 2018. ISSN 1558-9102. https://doi.org/10.1044/2018_JSLHR-H-17-0341. 2.2

[95] Dave F. Kleinschmidt and T. Florian Jaeger. Robust speech perception: Recognize the familiar, generalize to the similar, and adapt to the novel. Psychological Review, 122(2):148–203, April 2015. ISSN 1939-1471, 0033-295X. https://doi.org/10.1037/a0038695. 2.2

[96] R. E. Remez, P. E. Rubin, D. B. Pisoni, and T. D. Carrell. Speech perception without traditional speech cues. Science, 212(4497):947–949, May 1981. ISSN 0036-8075, 1095-9203. https://doi.org/10.1126/science.7233191. 2.2

[97] Matthew H. Davis, Ingrid S. Johnsrude, Alexis Hervais-Adelman, Karen Taylor, and Carolyn McGettigan. Lexical information drives perceptual learning of distorted speech: Evidence from the comprehension of noise-vocoded sentences. Journal of Experimental Psychology. General, 134(2):222–241, May 2005. ISSN 0096-3445. https://doi.org/10.1037/0096-3445.134.2.222. 2.2

[98] Taffeta M. Elliott and Frédéric E. Theunissen. The Modulation Transfer Function for Speech Intelligibility. PLOS Computational Biology, 5(3):e1000302, March 2009. ISSN 1553-7358. https://doi.org/10.1371/journal.pcbi.1000302. 2.2

[99] Justin J. Couchman, Mariana V. C. Coutinho, and J. David Smith. Rules and Resemblance: Their Changing Balance in the Category Learning of Humans (Homo sapiens) and Monkeys (Macaca mulatta). Journal of experimental psychology. Animal behavior processes, 36(2):172–183, April 2010. ISSN 0097-7403. https://doi.org/10.1037/a0016748. 2.2, 2.2

[100] Stephen E. G. Lea and A. J. Wills. Use of multiple dimensions in learned discriminations. Comparative Cognition & Behavior Reviews, 3, 2008. ISSN 19114745. https://doi.org/10.3819/ccbr.2008.30007. 2.2

[101] L. L. Holt, A. J. Lotto, and K. R. Kluender. Neighboring spectral content influences vowel identification. The Journal of the Acoustical Society of America, 108(2):710–722, August 2000. ISSN 0001-4966. https://doi.org/10.1121/1.429604. 2.2

[102] Lori L. Holt. The mean matters: Effects of statistically defined nonspeech spectral distributions on speech categorization. The Journal of the Acoustical Society of America, 120(5 Pt 1):2801–2817, November 2006. ISSN 0001-4966. https://doi.org/10.1121/1.2354071. 2.2

[103] Lori L. Holt. Temporally nonadjacent nonlinguistic sounds affect speech categorization. Psychological Science, 16(4):305–312, April 2005. ISSN 0956-7976. https://doi.org/10.1111/j.0956-7976.2005.01532.x. 2.2

[104] Patricia K Kuhl, Barbara T Conboy, Sharon Coffey-Corina, Denise Padden, Maritza Rivera-Gaxiola, and Tobey Nelson. Phonetic learning as a pathway to language: New data and native language magnet theory expanded (NLM-e). Philosophical Transactions of the Royal Society B: Biological Sciences, 363(1493):979–1000, March 2008. https://doi.org/10.1098/rstb.2007.2154. 2.2

[105] Foreign-language experience in infancy: Effects of short-term exposure and social interaction on phonetic learning | PNAS. https://www.pnas.org/content/100/15/9096. 2.2

[106] Iris van Rooij and Giosuè Baggio. Theory before the test: How to build high-verisimilitude explanatory theories in psychological science, February 2020. 2.2

[107] Patricia K. Kuhl. Brain Mechanisms in Early Language Acquisition. Neuron, 67(5):713–727, September 2010. ISSN 0896-6273. https://doi.org/10.1016/j.neuron.2010.08.038. 2.2

[108] Andrew J. King, Sundeep Teki, and Ben D. B. Willmore. Recent advances in understanding the auditory cortex. F1000Research, 7, 2018. ISSN 2046-1402. https://doi.org/10.12688/f1000research.15580.1. 2.2, 2.3

[109] Jennifer K. Schiavo and Robert C. Froemke. Capacities and neural mechanisms for auditory statistical learning across species. Hearing Research, 376:97–110, May 2019. ISSN 0378-5955. https://doi.org/10.1016/j.heares.2019.02.002. 2.2

[110] H B Barlow. Single Units and Sensation: A Neuron Doctrine for Perceptual Psychology? Perception, 1(4):371–394, December 1972. ISSN 0301-0066. https://doi.org/10.1068/p010371. 2.2

[111] Jennifer K. Bizley, Kerry M. M. Walker, Bernard W. Silverman, Andrew J. King, and Jan W. H. Schnupp. Interdependent encoding of pitch, timbre, and spatial location in auditory cortex. The Journal of Neuroscience: The Official Journal of the Society for Neuroscience, 29(7):2064–2075, February 2009. ISSN 1529-2401. https://doi.org/10.1523/JNEUROSCI.4755-08.2009. 2.2

[112] Kerry M. M. Walker, Jennifer K. Bizley, Andrew J. King, and Jan W. H. Schnupp. Multiplexed and robust representations of sound features in auditory cortex. The Journal of Neuroscience: The Official Journal of the Society for Neuroscience, 31(41):14565–14576, October 2011. ISSN 1529-2401. https://doi.org/10.1523/JNEUROSCI.2074-11.2011. 2.2

[113] Craig A. Atencio and Tatyana O. Sharpee. Multidimensional receptive field processing by cat primary auditory cortical neurons. Neuroscience, 359:130–141, September 2017. ISSN 1873-7544. https://doi.org/10.1016/j.neuroscience.2017.07.003. 2.2

[114] Tatyana O Sharpee, Craig A Atencio, and Christoph E Schreiner. Hierarchical representations in the auditory cortex. Current Opinion in Neurobiology, 21(5):761–767, October 2011. ISSN 0959-4388. https://doi.org/10.1016/j.conb.2011.05.027. 2.2

[115] Matthew V. Macellaio, Bing Liu, Jeffrey M. Beck, and Leslie C. Osborne. Why sensory neurons are tuned to multiple stimulus features. bioRxiv, page 2020.12.29.424235, December 2020. https://doi.org/10.1101/2020.12.29.424235. 2.2, 2.6

[116] C Angeloni and MN Geffen. Contextual modulation of sound processing in the auditory cortex. Current Opinion in Neurobiology, 49:8–15, April 2018. ISSN 0959-4388. https://doi.org/10.1016/j.conb.2017.10.012. 2.2

[117] Isabel Dean, Ben L. Robinson, Nicol S. Harper, and David McAlpine. Rapid Neural Adaptation to Sound Level Statistics. Journal of Neuroscience, 28(25):6430–6438, June 2008. ISSN 0270-6474, 1529-2401. https://doi.org/10.1523/JNEUROSCI.0470-08.2008. 2.2

[118] Neil C. Rabinowitz, Ben D. B. Willmore, Jan W. H. Schnupp, and Andrew J. King. Contrast gain control in auditory cortex. Neuron, 70(6):1178–1191, June 2011. ISSN 1097-4199. https://doi.org/10.1016/j.neuron.2011.04.030. 2.2

[119] Neil C. Rabinowitz, Ben D. B. Willmore, Andrew J. King, and Jan W. H. Schnupp. Constructing noise-invariant representations of sound in the auditory pathway. PLoS biology, 11(11):e1001710, November 2013. ISSN 1545-7885. https://doi.org/10.1371/journal.pbio.1001710. 2.2

[120] Nima Mesgarani, Stephen V. David, Jonathan B. Fritz, and Shihab A. Shamma. Mechanisms of noise robust representation of speech in primary auditory cortex. Proceedings of the National Academy of Sciences of the United States of America, 111(18):6792–6797, May 2014. ISSN 1091-6490. https://doi.org/10.1073/pnas.1318017111. 2.2

[121] Stephen V. David, Nima Mesgarani, Jonathan B. Fritz, and Shihab A. Shamma. Rapid synaptic depression explains nonlinear modulation of spectro-temporal tuning in primary auditory cortex by natural stimuli. The Journal of Neuroscience: The Official Journal of the Society for Neuroscience, 29(11):3374–3386, March 2009. ISSN 1529-2401. https://doi.org/10.1523/JNEUROSCI.5249-08.2009. 2.2

[122] Ryan G Natan, John J Briguglio, Laetitia Mwilambwe-Tshilobo, Sara I Jones, Mark Aizenberg, Ethan M Goldberg, and Maria Neimark Geffen. Complementary control of sensory adaptation by two types of cortical interneurons. eLife, 4: e09868, October 2015. ISSN 2050-084X. https://doi.org/10.7554/eLife.09868. 2.2

[123] Ryan G. Natan, Winnie Rao, and Maria N. Geffen. Cortical Interneurons Differentially Shape Frequency Tuning following Adaptation. Cell Reports, 21(4):878–890, October 2017. ISSN 2211-1247. https://doi.org/10.1016/j.celrep.2017.10.012. 2.2

[124] Jonathan Fritz, Shihab Shamma, Mounya Elhilali, and David Klein. Rapid task-related plasticity of spectrotemporal receptive fields in primary auditory cortex. Nature Neuroscience, 6(11):1216–1223, November 2003. ISSN 1097-6256. https://doi.org/10.1038/nn1141. 2.2

[125] Jonathan Fritz, Mounya Elhilali, and Shihab Shamma. Active listening: Task-dependent plasticity of spectrotemporal receptive fields in primary auditory cortex. Hearing Research, 206(1-2):159–176, August 2005. ISSN 0378-5955. https://doi.org/10.1016/j.heares.2005.01.015. 2.2

[126] Stephen V. David, Jonathan B. Fritz, and Shihab A. Shamma. Task reward structure shapes rapid receptive field plasticity in auditory cortex. Proceedings of the National Academy of Sciences of the United States of America, 109(6):2144–2149, February 2012. ISSN 1091-6490. https://doi.org/10.1073/pnas.1117717109. 2.2

[127] Stephen V. David and Shihab A. Shamma. Integration over multiple timescales in primary auditory cortex. The Journal of Neuroscience: The Official Journal of the Society for Neuroscience, 33(49):19154–19166, December 2013. ISSN 1529-2401. https://doi.org/10.1523/JNEUROSCI.2270-13.2013. 2.2

[128] D. B. Polley. Perceptual Learning Directs Auditory Cortical Map Reorganization through Top-Down Influences. Journal of Neuroscience, 26(18):4970–4982, 2006. ISSN 0270-6474. https://doi.org/10.1523/JNEUROSCI.3771-05.2006. 2.2

[129] Kasia M. Bieszczad and Norman M. Weinberger. Representational gain in cortical area underlies increase of memory strength. Proceedings of the National Academy of Sciences of the United States of America, 107(8):3793–3798, February 2010. ISSN 1091-6490. https://doi.org/10.1073/pnas.1000159107. 2.2

[130] Han Gyol Yi, Matthew K. Leonard, and Edward F. Chang. The Encoding of Speech Sounds in the Superior Temporal Gyrus. Neuron, 102(6):1096–1110, June 2019. ISSN 0896-6273. https://doi.org/10.1016/j.neuron.2019.04.023. 2.2

[131] N. Mesgarani, C. Cheung, K. Johnson, and E. F. Chang. Phonetic Feature Encoding in Human Superior Temporal Gyrus. Science, 343(6174):1006–1010, February 2014. ISSN 0036-8075, 1095-9203. https://doi.org/10.1126/science.1245994. 2.2

[132] Edward F. Chang, Jochem W. Rieger, Keith Johnson, Mitchel S. Berger, Nicholas M. Barbaro, and Robert T. Knight. Categorical speech representation in human superior temporal gyrus. Nature Neuroscience, 13(11):1428–1432, November 2010. ISSN 1546-1726. https://doi.org/10.1038/nn.2641. 2.2

[133] Alexander M. Chan, Andrew R. Dykstra, Vinay Jayaram, Matthew K. Leonard, Katherine E. Travis, Brian Gygi, Janet M. Baker, Emad Eskandar, Leigh R. Hochberg, Eric Halgren, and Sydney S. Cash. Speech-Specific Tuning of Neurons in Human Superior Temporal Gyrus. Cerebral Cortex, 24(10):2679–2693, October 2014. ISSN 1047-3211. https://doi.org/10.1093/cercor/bht127. 2.2

[134] Liberty S. Hamilton, Erik Edwards, and Edward F. Chang. A Spatial Map of Onset and Sustained Responses to Speech in the Human Superior Temporal Gyrus. Current Biology, 28(12):1860–1871.e4, June 2018. ISSN 0960-9822. https://doi.org/10.1016/j.cub.2018.04.033. 2.2, 2.2

[135] Crystal T. Engineer, Kimiya C. Rahebi, Elizabeth P. Buell, Melyssa K. Fink, and Michael P. Kilgard. Speech training alters consonant and vowel responses in multiple auditory cortex fields. Behavioural Brain Research, 287:256–264, July 2015. ISSN 0166-4328. https://doi.org/10.1016/j.bbr.2015.03.044. 2.2

[136] Jonny L. Saunders and Michael Wehr. Mice can learn phonetic categories. The Journal of the Acoustical Society of America, 145(3):1168–1177, March 2019. ISSN 0001-4966. https://doi.org/10.1121/1.5091776. 2.2, 4.4

[137] H. Takahashi, R. Yokota, A. Funamizu, H. Kose, and R. Kanzaki. Learning-stage-dependent, field-specific, map plasticity in the rat auditory cortex during appetitive operant conditioning. Neuroscience, 199:243–258, December 2011. ISSN 0306-4522. https://doi.org/10.1016/j.neuroscience.2011.09.046. 2.2

[138] Zhiyue Shi, Sumei Yan, Yu Ding, Chang Zhou, Shaowen Qian, Zhaoqun Wang, Chen Gong, Meng Zhang, Yanjie Zhang, Yandong Zhao, Huizhong Wen, Penghui Chen, Qiyue Deng, Tiantian Luo, Ying Xiong, and Yi Zhou. Anterior Auditory Field Is Needed for Sound Categorization in Fear Conditioning Task of Adult Rat. Frontiers in Neuroscience, 13, 2019. ISSN 1662-453X. https://doi.org/10.3389/fnins.2019.01374. 2.2

[139] Andres Carrasco and Stephen G. Lomber. Neuronal activation times to simple, complex, and natural sounds in cat primary and nonprimary auditory cortex. Journal of Neurophysiology, 106(3):1166–1178, June 2011. ISSN 0022-3077. https://doi.org/10.1152/jn.00940.2010. 2.2

[140] Jennifer F. Linden, Robert C. Liu, Maneesh Sahani, Christoph E. Schreiner, and Michael M. Merzenich. Spectrotemporal Structure of Receptive Fields in Areas AI and AAF of Mouse Auditory Cortex. Journal of Neurophysiology, 90(4):2660–2675, October 2003. ISSN 0022-3077. https://doi.org/10.1152/jn.00751.2002. 2.2

[141] Pritesh K. Pandya, Daniel L. Rathbun, Raluca Moucha, Navzer D. Engineer, and Michael P. Kilgard. Spectral and Temporal Processing in Rat Posterior Auditory Cortex. Cerebral Cortex, 18(2):301–314, February 2008. ISSN 1047-3211. https://doi.org/10.1093/cercor/bhm055. 2.2

[142] Andres Carrasco and Stephen G. Lomber. Evidence for Hierarchical Processing in Cat Auditory Cortex: Nonreciprocal Influence of Primary Auditory Cortex on the Posterior Auditory Field. Journal of Neuroscience, 29(45):14323–14333, November 2009. ISSN 0270-6474, 1529-2401. https://doi.org/10.1523/JNEUROSCI.2905-09.2009. 2.2

[143] Xiaoqin Wang. Neural coding strategies in auditory cortex. Hearing Research, 229(1):81–93, July 2007. ISSN 0378-5955. https://doi.org/10.1016/j.heares.2007.01.019. 2.2

[144] Zoe C. Ashwood, Nicholas A. Roy, Iris R. Stone, The International Brain Laboratory, Anne K. Churchland, Alexandre Pouget, and Jonathan W. Pillow. Mice alternate between discrete strategies during perceptual decision-making. Preprint, Neuroscience, October 2020. 2.3, 2.6

[145] Sam V. Norman-Haignere, Laura K. Long, Orrin Devinsky, Werner Doyle, Ifeoma Irobunda, Edward M. Merricks, Neil A. Feldstein, Guy M. McKhann, Catherine A. Schevon, Adeen Flinker, and Nima Mesgarani. Hierarchical integration across multiple timescales in human auditory cortex. bioRxiv, page 2020.09.30.321687, October 2020. https://doi.org/10.1101/2020.09.30.321687. 2.3, 2.6

[146] Fernando E. Rosas, Pedro A. M. Mediano, Martin Biehl, Shamil Chandaria, and Daniel Polani. Causal blankets: Theory and algorithmic framework. arXiv:2008.12568 [nlin, q-bio], September 2020. 2.3, 2.6

[147] Alexis Dubreuil, Adrian Valente, Manuel Beiran, Francesca Mastrogiuseppe, and Srdjan Ostojic. Complementary roles of dimensionality and population structure in neural computations. bioRxiv, page 2020.07.03.185942, July 2020. https://doi.org/10.1101/2020.07.03.185942. 2.3, 2.6

[148] Chethan Pandarinath, Daniel J. O'Shea, Jasmine Collins, Rafal Jozefowicz, Sergey D. Stavisky, Jonathan C. Kao, Eric M. Trautmann, Matthew T. Kaufman, Stephen I. Ryu, Leigh R. Hochberg, Jaimie M. Henderson, Krishna V. Shenoy, L. F. Abbott, and David Sussillo. Inferring single-trial neural population dynamics using sequential auto-encoders. Nature Methods, 15(10):805–815, October 2018. ISSN 1548-7105. https://doi.org/10.1038/s41592-018-0109-9. 2.3

[149] Björn Brembs. The brain as a dynamically active organ. Biochemical and Biophysical Research Communications, December 2020. ISSN 0006-291X. https://doi.org/10.1016/j.bbrc.2020.12.011. 2.3

[150] Juan A. Gallego, Matthew G. Perich, Stephanie N. Naufel, Christian Ethier, Sara A. Solla, and Lee E. Miller. Cortical population activity within a preserved neural manifold underlies multiple motor behaviors. Nature Communications, 9 (1):4233, October 2018. ISSN 2041-1723. https://doi.org/10.1038/s41467-018-06560-z. 2.3, 2.6, 2.6

[151] Eleanor Rosch. Wittgenstein and Categorization Research in Cognitive Psychology. In Michael Chapman and Roger A. Dixon, editors, Meaning and the Growth of Understanding: Wittgenstein's Significance for Developmental Psychology, pages 151–166. Springer, Berlin, Heidelberg, 1987. ISBN 978-3-642-83023-5. https://doi.org/10.1007/978-3-642-83023-5_9. 2.3

[152] R. C. deCharms, D. T. Blake, and M. M. Merzenich. Optimizing sound features for cortical neurons. Science (New York, N.Y.), 280(5368):1439–1443, May 1998. ISSN 0036-8075. https://doi.org/10.1126/science.280.5368.1439. 2.3

[153] Misha B. Ahrens, Jennifer F. Linden, and Maneesh Sahani. Nonlinearities and contextual influences in auditory cortical responses modeled with multilinear spectrotemporal methods. The Journal of Neuroscience: The Official Journal of the Society for Neuroscience, 28(8):1929–1942, February 2008. ISSN 1529-2401. https://doi.org/10.1523/JNEUROSCI.3377-07.2008. 2.3

[154] Friedrich Schuessler, Francesca Mastrogiuseppe, Alexis Dubreuil, Srdjan Ostojic, and Omri Barak. The interplay between randomness and structure during learning in RNNs. arXiv:2006.11036 [q-bio], October 2020. 2.3

[155] David H Krantz and Amos Tversky. Similarity of rectangles: An analysis of subjective dimensions. Journal of Mathematical Psychology, 12(1):4–34, February 1975. ISSN 00222496. https://doi.org/10.1016/0022-2496(75)90047-4. 2.3

[156] Daniel Durstewitz, Nicole M. Vittoz, Stan B. Floresco, and Jeremy K. Seamans. Abrupt Transitions between Prefrontal Neural Ensemble States Accompany Behavioral Transitions during Rule Learning. Neuron, 66(3):438–448, May 2010. ISSN 0896-6273. https://doi.org/10.1016/j.neuron.2010.03.029. 2.6, 2.6

[157] Matthew G. Perich, Juan A. Gallego, and Lee E. Miller. A Neural Population Mechanism for Rapid Learning. Neuron, 100(4):964–976.e7, November 2018. ISSN 0896-6273. https://doi.org/10.1016/j.neuron.2018.09.030. 2.6

[158] Mark R. Saddler, Ray Gonzalez, and Josh H. McDermott. Deep neural network models reveal interplay of peripheral coding and stimulus statistics in pitch perception. Preprint, Animal Behavior and Cognition, November 2020. 2.6

[159] Francesca Mastrogiuseppe and Srdjan Ostojic. Linking Connectivity, Dynamics, and Computations in Low-Rank Recurrent Neural Networks. Neuron, 99(3):609–623.e29, August 2018. ISSN 0896-6273. https://doi.org/10.1016/j.neuron.2018.07.003. 2.6

[160] Greta Kaufeld, Hans Rutger Bosker, Sanne ten Oever, Phillip M. Alday, Antje S. Meyer, and Andrea E. Martin. Linguistic Structure and Meaning Organize Neural Oscillations into a Content-Specific Hierarchy. Journal of Neuroscience, 40(49): 9467–9475, December 2020. ISSN 0270-6474, 1529-2401. https://doi.org/10.1523/JNEUROSCI.0302-20.2020. 2.6

[161] Manuel Beiran, Alexis Dubreuil, Adrian Valente, Francesca Mastrogiuseppe, and Srdjan Ostojic. Shaping dynamics with multiple populations in low-rank recurrent networks. arXiv:2007.02062 [q-bio], November 2020. 2.6

[162] Ines Hipolito, Maxwell Ramstead, Laura Convertino, Anjali Bhat, Karl Friston, and Thomas Parr. Markov Blankets in the Brain. arXiv:2006.02741 [physics, q-bio], June 2020. 2.6

[163] Philip R. L. Parker, Morgan A. Brown, Matthew C. Smear, and Cristopher M. Niell. Movement-Related Signals in Sensory Areas: Roles in Natural Behavior. Trends in Neurosciences, 43(8):581–595, August 2020. ISSN 0166-2236, 1878-108X. https://doi.org/10.1016/j.tins.2020.05.005. 2.6

[164] Matthew Warburton, Jack Brookes, Mohamed Hasan, Matteo Leonetti, Mehmet Dogar, He Wang, Anthony G. Cohn, Faisal Mushtaq, and Mark A. Mon-Williams. Getting stuck in a rut as an emergent feature of a dynamic decision-making system. bioRxiv, page 2020.06.02.127860, June 2020. https://doi.org/10.1101/2020.06.02.127860. 2.6

[165] Sukbin Lim, Jillian L. McKee, Luke Woloszyn, Yali Amit, David J. Freedman, David L. Sheinberg, and Nicolas Brunel. Inferring learning rules from distributions of firing rates in cortical neurons. Nature Neuroscience, 18(12):1804–1810, December 2015. ISSN 1546-1726. https://doi.org/10.1038/nn.4158. 2.6

[166] Federico Battiston, Giulia Cencetti, Iacopo Iacopini, Vito Latora, Maxime Lucas, Alice Patania, Jean-Gabriel Young, and Giovanni Petri. Networks beyond pairwise interactions: Structure and dynamics. arXiv:2006.01764 [cond-mat, physics:nlin, physics:physics, q-bio], June 2020. 2.6

[167] Hiroyuki K. Kato, Monica W. Chu, Jeffry S. Isaacson, and Takaki Komiyama. Dynamic Sensory Representations in the Olfactory Bulb: Modulation by Wakefulness and Experience. Neuron, 76(5):962–975, December 2012. ISSN 0896-6273. https://doi.org/10.1016/j.neuron.2012.09.037. 2.6

[168] Juan A. Gallego, Matthew G. Perich, Lee E. Miller, and Sara A. Solla. Neural Manifolds for the Control of Movement. Neuron, 94(5):978–984, June 2017. ISSN 1097-4199. https://doi.org/10.1016/j.neuron.2017.05.025. 2.6

[169] Tony Hyun Kim, Yanping Zhang, Jérôme Lecoq, Juergen C. Jung, Jane Li, Hongkui Zeng, Cristopher M. Niell, and Mark J. Schnitzer. Long-Term Optical Access to an Estimated One Million Neurons in the Live Mouse Cortex. Cell Reports, 17(12):3385–3394, December 2016. ISSN 2211-1247. https://doi.org/10.1016/j.celrep.2016.12.004. 2.6

[170] Robert B. Levy, Tiemo Marquarding, Ashlan P. Reid, Christopher M. Pun, Nicolas Renier, and Hysell V. Oviedo. Circuit asymmetries underlie functional lateralization in the mouse auditory cortex. Nature Communications, 10(1):2783, June 2019. ISSN 2041-1723. https://doi.org/10.1038/s41467-019-10690-3. 2.6

[171] Sung-Joo Lim, Julie A. Fiez, and Lori L. Holt. How may the basal ganglia contribute to auditory categorization and speech perception? Frontiers in Neuroscience, 8, 2014. ISSN 1662-453X. https://doi.org/10.3389/fnins.2014.00230. 2.6

[172] Josef P Rauschecker and Sophie K Scott. Maps and streams in the auditory cortex: Nonhuman primates illuminate human speech processing. Nature neuroscience, 12(6):718–724, June 2009. ISSN 1097-6256. https://doi.org/10.1038/nn.2331. 2.6

[173] Meghan Clayards. Differences in cue weights for speech perception are correlated for individuals within and across contrasts. The Journal of the Acoustical Society of America, 144(3):EL172–EL177, September 2018. ISSN 0001-4966. https://doi.org/10.1121/1.5052025. 2.6

[174] Isaac M. Carruthers, Diego A. Laplagne, Andrew Jaegle, John J. Briguglio, Laetitia Mwilambwe-Tshilobo, Ryan G. Natan, and Maria N. Geffen. Emergence of invariant representation of vocalizations in the auditory cortex. Journal of Neurophysiology, 114(5):2726–2740, August 2015. ISSN 0022-3077. https://doi.org/10.1152/jn.00095.2015. 2.6

[175] Philip Meier, Erik Flister, and Pamela Reinagel. Collinear features impair visual detection by rats. Journal of Vision, 11 (3), March 2011. ISSN 1534-7362. https://doi.org/10.1167/11.3.22. II

[176] Santiago Jaramillo, Anna Lakunina, Lan Guo, and Nick Ponvert. TASKontrol, January 2022. II

[177] Jacob Reimer, Matthew J. McGinley, Yang Liu, Charles Rodenkirch, Qi Wang, David A. McCormick, and Andreas S. Tolias. Pupil fluctuations track rapid changes in adrenergic and cholinergic activity in cortex. Nature Communications, 7:13289, November 2016. ISSN 2041-1723. https://doi.org/10.1038/ncomms13289. 3

[178] Ana Parabucki, Alexander Bizer, Genela Morris, Antonio E. Munoz, Avinash D. S. Bala, Matthew Smear, and Roman Shusterman. Odor Concentration Change Coding in the Olfactory Bulb. eNeuro, 6(1), February 2019. ISSN 2373-2822. https://doi.org/10.1523/ENEURO.0396-18.2019. 3

[179] Cristopher M. Niell and Michael P. Stryker. Modulation of Visual Responses by Behavioral State in Mouse Visual Cortex. Neuron, 65(4):472–479, February 2010. ISSN 0896-6273. https://doi.org/10.1016/j.neuron.2010.01.033. 3

[180] Tanmay Nath, Alexander Mathis, An Chi Chen, Amir Patel, Matthias Bethge, and Mackenzie Weygandt Mathis. Using DeepLabCut for 3D markerless pose estimation across species and behaviors. Nature Protocols, 14(7):2152–2176, July 2019. ISSN 1750-2799. https://doi.org/10.1038/s41596-019-0176-0. 3

[181] James J. Jun, Nicholas A. Steinmetz, Joshua H. Siegle, Daniel J. Denman, Marius Bauza, Brian Barbarits, Albert K. Lee, Costas A. Anastassiou, Alexandru Andrei, Çağatay Aydın, Mladen Barbic, Timothy J. Blanche, Vincent Bonin, João Couto, Barundeb Dutta, Sergey L. Gratiy, Diego A. Gutnisky, Michael Häusser, Bill Karsh, Peter Ledochowitsch, Carolina Mora Lopez, Catalin Mitelut, Silke Musa, Michael Okun, Marius Pachitariu, Jan Putzeys, P. Dylan Rich, Cyrille Rossant, Wei-Lung Sun, Karel Svoboda, Matteo Carandini, Kenneth D. Harris, Christof Koch, John O'Keefe, and Timothy D. Harris. Fully integrated silicon probes for high-density recording of neural activity. Nature, 551(7679):232–236, November 2017. ISSN 1476-4687. https://doi.org/10.1038/nature24636. 3, 4.8, 11

[182] Christopher P. Burgess, Armin Lak, Nicholas A. Steinmetz, Peter Zatka-Haas, Charu Bai Reddy, Elina A. K. Jacobs, Jennifer F. Linden, Joseph J. Paton, Adam Ranson, Sylvia Schröder, Sofia Soares, Miles J. Wells, Lauren E. Wool, Kenneth D. Harris, and Matteo Carandini. High-Yield Methods for Accurate Two-Alternative Visual Psychophysics in Head-Fixed Mice. Cell Reports, 20(10):2513–2524, September 2017. ISSN 2211-1247. https://doi.org/10.1016/j.celrep.2017.08.047. 3

[183] Kay Thurley and Aslı Ayaz. Virtual reality systems for rodents. Current Zoology, 63(1):109–119, February 2017. ISSN 1674-5507. https://doi.org/10.1093/cz/zow070. 3

[184] Anna R. Chambers, Kenneth E. Hancock, Kamal Sen, and Daniel B. Polley. Online stimulus optimization rapidly reveals multidimensional selectivity in auditory cortical neurons. The Journal of Neuroscience, 34(27):8963–8975, July 2014. ISSN 1529-2401. https://doi.org/10.1523/JNEUROSCI.0260-14.2014. 3

[185] Chance Elliott, Vipin Vijayakumar, Wesley Zink, and Richard Hansen. National Instruments LabVIEW: A Programming Environment for Laboratory Automation and Measurement. JALA: Journal of the Association for Laboratory Automation, 12(1):17–24, February 2007. ISSN 1535-5535. https://doi.org/10.1016/j.jala.2006.07.012. 3

[186] Open Ephys. pyControl. http://www.open-ephys.org/store/pycontrol, February 2019. 3

[187] Josh Sanders. Sanworks - BPod. https://www.sanworks.io/shop/products.php?productFamily=bpod. 3

[188] Matthew B. Wall. Reliability starts with the experimental tools employed. Cortex, 113:352–354, April 2019. ISSN 0010-9452. https://doi.org/10.1016/j.cortex.2018.11.034. 3, 4.3

[189] Peter Johnson-Lenz and Trudy Johnson-Lenz. Post-mechanistic groupware primitives: Rhythms, boundaries and containers. International Journal of Man-Machine Studies, 34(3):395–417, February 1991. ISSN 0020-7373. https://doi.org/10.1016/0020-7373(91)90027-5. 3, 1

[190] Peter Johnson-Lenz and Trudy Johnson-Lenz. Groupware: Coining and defining it. ACM SIGGROUP Bulletin, 19(2): 34, August 1998. ISSN 2372-7403, 2372-739X. https://doi.org/10.1145/290575.290585. 3

[191] STEPHEN R. BARLEY and BETH A. BECHKY. In the Backrooms of Science: The Work of Technicians in Science Labs. Work and Occupations, 21(1):85–126, February 1994. ISSN 0730-8884. https://doi.org/10.1177/073088849402100104. 3

[192] Thomas Akam, Andy Lustig, James M Rowland, Sampath KT Kapanaiah, Joan Esteve-Agraz, Mariangela Panniello, Cristina Márquez, Michael M Kohl, Dennis Kätzel, Rui M Costa, and Mark E Walton. Open-source, Python-based, hardware and software for controlling behavioural neuroscience experiments. eLife, 11:e67846, January 2022. ISSN 2050-084X. https://doi.org/10.7554/eLife.67846. 3.1

[193] Gonçalo Lopes, Niccolò Bonacchi, João Frazão, Joana P. Neto, Bassam V. Atallah, Sofia Soares, Luís Moreira, Sara Matias, Pavel M. Itskov, Patrícia A. Correia, Roberto E. Medina, Lorenza Calcaterra, Elena Dreosti, Joseph J. Paton, and Adam R. Kampff. Bonsai: An event-based framework for processing and controlling data streams. Frontiers in Neuroinformatics, 9, 2015. ISSN 1662-5196. 5, 7.3

[194] Florian Krause and Oliver Lindemann. Expyriment: A Python library for cognitive and neuroscientific experiments. Behavior Research Methods, 46(2):416–428, June 2014. ISSN 1554-3528. https://doi.org/10.3758/s13428-013-0390-6. 5

[195] Jonathan Peirce, Jeremy R. Gray, Sol Simpson, Michael MacAskill, Richard Höchenberger, Hiroyuki Sogo, and Erik Kastman. PsychoPy2: Experiments in behavior made easy. Behavior Research Methods, 51(1):195–203, February 2019. ISSN 1554-3528. https://doi.org/10.3758/s13428-018-01193-y. 5, 5.6

[196] Sebastiaan Mathôt, Daniel Schreij, and Jan Theeuwes. OpenSesame: An open-source, graphical experiment builder for the social sciences. Behavior Research Methods, 44(2):314–324, June 2012. ISSN 1554-3528. https://doi.org/10.3758/s13428-011-0168-7. 5

[197] Xinfeng Chen and Haohong Li. ArControl: An Arduino-Based Comprehensive Behavioral Platform with Real-Time Performance. Frontiers in Behavioral Neuroscience, 11, 2017. ISSN 1662-5153. https://doi.org/10.3389/fnbeh.2017.00244. 5

[198] S. van der Walt, S. C. Colbert, and G. Varoquaux. The NumPy Array: A Structure for Efficient Numerical Computation. Computing in Science Engineering, 13(2):22–30, March 2011. ISSN 1521-9615. https://doi.org/10.1109/MCSE.2011.37. 6, 6.1

[199] Eric Jones, Travis Oliphant, and Pearu Peterson. SciPy: Open Source Scientific Tools for Python. http://www.scipy.org/, 2001. 6

[200] Sandeep Robert Datta, David J. Anderson, Kristin Branson, Pietro Perona, and Andrew Leifer. Computational Neuroethology: A Call to Action. Neuron, 104(1):11–24, October 2019. ISSN 0896-6273. https://doi.org/10.1016/j.neuron.2019.09.038. 3.2, 5.7

[201] Sanworks, LLC. 8 reasons to use Bpod's new HiFi module. https://sanworks.io/news/viewArticle.php?articleID=HiFi01, July 2021. 3.2

[202] The International Brain Laboratory, Valeria Aguillon-Rodriguez, Dora E. Angelaki, Hannah M. Bayer, Niccolò Bonacchi, Matteo Carandini, Fanny Cazettes, Gaelle A. Chapuis, Anne K. Churchland, Yang Dan, Eric E. J. Dewitt, Mayo Faulkner, Hamish Forrest, Laura M. Haetzel, Michael Hausser, Sonja B. Hofer, Fei Hu, Anup Khanal, Christopher S. Krasniak, Inês Laranjeira, Zachary F. Mainen, Guido T. Meijer, Nathaniel J. Miska, Thomas D. Mrsic-Flogel, Masayoshi Murakami, Jean-Paul Noel, Alejandro Pan-Vazquez, Cyrille Rossant, Joshua I. Sanders, Karolina Z. Socha, Rebecca Terry, Anne E. Urai, Hernando M. Vergara, Miles J. Wells, Christian J. Wilson, Ilana B. Witten, Lauren E. Wool, and Anthony Zador. Standardized and reproducible measurement of decision-making in mice, October 2020. 3.2

[203] Tim Anderson. Guido van Rossum aiming to make CPython 2x faster in 3.11. https://www.theregister.com/2021/05/13/guido_van_rossum_cpython_3_11/, May 2021. 2

[204] Guido van Rossum. Glue It All Together With Python. https://www.python.org/doc/essays/omg-darpa-mcc-position/, January 1998. 4.1

[205] Gary A Kane, Gonçalo Lopes, Jonny L Saunders, Alexander Mathis, and Mackenzie W Mathis. Real-time, low-latency closed-loop feedback using markerless posture tracking. eLife, 9:e61909, December 2020. ISSN 2050-084X. https://doi.org/10.7554/eLife.61909. 4.2, 5.5, 5.5, 5.6, 15

[206] Pieter Hintjens. ZeroMQ: Messaging for Many Applications. O'Reilly Media, Beijing, 1 edition edition, March 2013. ISBN 978-1-4493-3406-2. 4.2, 9

[207] Oliver Rübel, Andrew Tritt, Benjamin Dichter, Thomas Braun, Nicholas Cain, Nathan Clack, Thomas J. Davidson, Max Dougherty, Jean-Christophe Fillion-Robin, Nile Graddis, Michael Grauer, Justin T. Kiggins, Lawrence Niu, Doruk Ozturk, William Schroeder, Ivan Soltesz, Friedrich T. Sommer, Karel Svoboda, Ng Lydia, Loren M. Frank, and Kristofer Bouchard. NWB:N 2.0: An Accessible Data Standard for Neurophysiology. bioRxiv, January 2019. https://doi.org/10.1101/523035. 4.3, 7.4

[208] David A. W. Soergel. Rampant software errors may undermine scientific results. F1000Research, 3, July 2015. ISSN 2046-1402. https://doi.org/10.12688/f1000research.5930.2. 4.3

[209] Anders Eklund, Thomas E. Nichols, and Hans Knutsson. Cluster failure: Why fMRI inferences for spatial extent have inflated false-positive rates. Proceedings of the National Academy of Sciences, 113(28):7900–7905, July 2016. ISSN 0027-8424, 1091-6490. https://doi.org/10.1073/pnas.1602413113. 4.3

[210] Jayanti Bhandari Neupane, Ram P. Neupane, Yuheng Luo, Wesley Y. Yoshida, Rui Sun, and Philip G. Williams. Characterization of Leptazolines A–D, Polar Oxazolines from the Cyanobacterium Leptolyngbya sp., Reveals a Glitch with the "Willoughby–Hoye" Scripts for Calculating NMR Chemical Shifts. Organic Letters, 21(20):8449–8453, October 2019. ISSN 1523-7060. https://doi.org/10.1021/acs.orglett.9b03216. 4.3

[211] Greg Miller. A Scientist's Nightmare: Software Problem Leads to Five Retractions. Science, 314(5807):1856–1857, December 2006. ISSN 0036-8075, 1095-9203. https://doi.org/10.1126/science.314.5807.1856. 4.3

[212] Yarden Katz and Ulrich Bernhard Matter. On the Biomedical Elite: Inequality and Stasis in Scientific Knowledge Production. Berkman Klein Center for Internet & Society Research, July 2017. 4.3

[213] Jeremy Ashkenas, Haeyoun Park, and Adam Pearce. Even With Affirmative Action, Blacks and Hispanics Are More Underrepresented at Top Colleges Than 35 Years Ago. The New York Times, August 2017. ISSN 0362-4331. 4.3

[214] Aaron Clauset, Samuel Arbesman, and Daniel B. Larremore. Systematic inequality and hierarchy in faculty hiring networks. Science Advances, 1(1):e1400005, February 2015. ISSN 2375-2548. https://doi.org/10.1126/sciadv.1400005. 4.3

[215] J. M. Pearce, J. C. Molloy, S. Kuznetsov, and S. Dosemagen. Expanding Equitable Access to Experimental Research and STEM Education by Supporting Open Source Hardware Development, January 2019. 4.3

[216] Emmeke Aarts, Matthijs Verhage, Jesse V. Veenvliet, Conor V. Dolan, and Sophie van der Sluis. A solution to dependency: Using multilevel analysis to accommodate nested data. Nature Neuroscience, 17(4):491–496, April 2014. ISSN 1546-1726. https://doi.org/10.1038/nn.3648. 4.8, 4.3

[217] Patrick E. Shrout and Joseph L. Rodgers. Psychology, Science, and Knowledge Construction: Broadening Perspectives from the Replication Crisis. Annual Review of Psychology, 69(1):487–510, 2018. https://doi.org/10.1146/annurev-psych-122216-011845. 4.3

[218] Katherine S. Button, John P. A. Ioannidis, Claire Mokrysz, Brian A. Nosek, Jonathan Flint, Emma S. J. Robinson, and Marcus R. Munafò. Power failure: Why small sample size undermines the reliability of neuroscience. Nature Reviews Neuroscience, 14(5):365–376, May 2013. ISSN 1471-0048. https://doi.org/10.1038/nrn3475. 4.3

[219] Wes McKinney. Pandas: A foundational Python library for data analysis and statistics. Python for high performance and scientific computing, 14(9):1–9, 2011. 5.2, 6.1

[220] Dexter C. Kozen. Limitations of Finite Automata. In Dexter C. Kozen, editor, Automata and Computability, Undergraduate Texts in Computer Science, pages 67–71. Springer New York, New York, NY, 1997. ISBN 978-1-4612-1844-9. https://doi.org/10.1007/978-1-4612-1844-9_12. 5.3

[221] Ji Hyun Bak, Jung Yoon Choi, Athena Akrami, Ilana Witten, and Jonathan W Pillow. Adaptive optimal training of animal behavior. In D. D. Lee, M. Sugiyama, U. V. Luxburg, I. Guyon, and R. Garnett, editors, Advances in Neural Information Processing Systems 29, pages 1947–1955. Curran Associates, Inc., 2016. 5.3

[222] Aaron Swartz. Aaron Swartz's A Programmable Web: An Unfinished Work. Synthesis Lectures on the Semantic Web: Theory and Technology, 3(2):1–64, February 2013. ISSN 2160-4711, 2160-472X. https://doi.org/10.2200/S00481ED1V01Y201302WBE005. 4

[223] Fatemeh Abyarjoo, Armando Barreto, Jonathan Cofino, and Francisco R. Ortega. Implementing a Sensor Fusion Algorithm for 3D Orientation Detection with Inertial/Magnetic Sensors. Innovations and Advances in Computing, Informatics, Systems Sciences, Networking and Engineering, pages 305–310, 2015. https://doi.org/10.1007/978-3-319-06773-5_41. 5.5

[224] Photis Patonis, Petros Patias, Ilias N. Tziavos, Dimitrios Rossikopoulos, and Konstantinos G. Margaritis. A Fusion Method for Combining Low-Cost IMU/Magnetometer Outputs for Use in Applications on Mobile Devices. Sensors (Basel, Switzerland), 18(8), August 2018. ISSN 1424-8220. https://doi.org/10.3390/s18082616. 5.5

[225] Leland Wilkinson. The Grammar of Graphics. In James E. Gentle, Wolfgang Karl Härdle, and Yuichi Mori, editors, Handbook of Computational Statistics: Concepts and Methods, Springer Handbooks of Computational Statistics, pages 375–414. Springer Berlin Heidelberg, Berlin, Heidelberg, 2012. ISBN 978-3-642-21551-3. 5.9

[226] Hadley Wickham. Ggplot2: Elegant Graphics for Data Analysis. Springer-Verlag New York, June 2016. ISBN 978-3-319-24277-4. 6.1

[227] Hadley Wickham, Romain François, Lionel Henry, and Kirill Müller. Dplyr: A Grammar of Data Manipulation, 2022. 6.1

[228] Lionel Henry and Hadley Wickham. Purrr: Functional Programming Tools, 2020. 6.1

[229] Charles R. Harris, K. Jarrod Millman, Stéfan J. van der Walt, Ralf Gommers, Pauli Virtanen, David Cournapeau, Eric Wieser, Julian Taylor, Sebastian Berg, Nathaniel J. Smith, Robert Kern, Matti Picus, Stephan Hoyer, Marten H. van Kerkwijk, Matthew Brett, Allan Haldane, Jaime Fernández del Río, Mark Wiebe, Pearu Peterson, Pierre Gérard-Marchant, Kevin Sheppard, Tyler Reddy, Warren Weckesser, Hameer Abbasi, Christoph Gohlke, and Travis E. Oliphant. Array programming with NumPy. Nature, 585(7825):357–362, September 2020. ISSN 1476-4687. https://doi.org/10.1038/s41586-020-2649-2. 6.1

[230] Eduardo Soares, Pedro Brandão, and Rui Prior. Analysis of Timekeeping in Experimentation. In 2020 12th International Symposium on Communication Systems, Networks and Digital Signal Processing (CSNDSP), pages 1–6, July 2020. https://doi.org/10.1109/CSNDSP49049.2020.9249632. 6.3, 3

[231] Joshua H. Siegle, Aarón Cuevas López, Yogi A. Patel, Kirill Abramov, Shay Ohayon, and Jakob Voigts. Open Ephys: An open-source, plugin-based platform for multichannel electrophysiology. Journal of Neural Engineering, 14(4):045003, June 2017. ISSN 1741-2552. https://doi.org/10.1088/1741-2552/aa5eea. 7.3

[232] Daniel Aharoni and Tycho M. Hoogland. Circuit Investigations With Open-Source Miniaturized Microscopes: Past, Present and Future. Frontiers in Cellular Neuroscience, 13, 2019. ISSN 1662-5102. 7.3

[233] Daniel Aharoni, Baljit S. Khakh, Alcino J. Silva, and Peyman Golshani. All the light that we can see: A new era in miniaturized microscopy. Nature Methods, 16(1):11–13, January 2019. ISSN 1548-7105. https://doi.org/10.1038/s41592-018-0266-x. 7.3

[234] Dimitri Yatsenko, Edgar Y. Walker, and Andreas S. Tolias. DataJoint: A Simpler Relational Data Model, July 2018. 7.4