Supplementary Appendix

This appendix has been provided by the authors to give readers additional information about their work.

Supplement to: Moses DA, Metzger SL, Liu JR, et al. Neuroprosthesis for decoding speech in a paralyzed person with anarthria. N Engl J Med 2021;385:217-27. DOI: 10.1056/NEJMoa2027540

Contents

List of investigators and contributions

List of investigators (authors)

1. David A. Moses*
2. Sean L. Metzger*
3. Jessie R. Liu*
4. Gopala K. Anumanchipalli
5. Joseph G. Makin
6. Pengfei F. Sun
7. Josh Chartier
8. Maximilian E. Dougherty
9. Patricia M. Liu
10. Gary M. Abrams
11. Adelyn Tu-Chan
12. Karunesh Ganguly
13. Edward F. Chang

*These three authors contributed equally

Author contributions

D.A.M. led the research project, designed the language model and Viterbi decoder, and developed the real-time software infrastructure. S.L.M. developed the real-time word classification models. J.R.L. developed the real-time speech detection models. D.A.M., S.L.M., and J.R.L. analyzed the data, and, along with E.F.C., wrote and submitted the manuscript (with input from other authors). D.A.M., S.L.M., J.R.L., G.K.A., J.G.M., P.F.S., J.C., and M.E.D. participated in data collection and methodological design. M.E.D., K.G., A. T.-C., and E.F.C. performed regulatory and clinical supervision. P.M.L. provided speech-language pathology consultation. G.M.A. helped with recruitment. E.F.C. conceived, designed, and supervised the clinical trial.

Supplementary sections

Section S1. The participant's assistive typing device

Assistive typing device description

The participant often uses a commercially available touch-screen typing interface (Tobii Dynavox) to communicate with others, which he controls with a long (approximately 18-inch) plastic stylus attached to a baseball cap by using residual head and neck movement. The device displays letters, words, and other options (such as punctuation) that the participant can select with his stylus, enabling him to construct a text string. After creating the desired text string, the participant can use his stylus to press an icon that synthesizes the text string into an audible speech waveform. This process of spelling out a desired message and having the device synthesize it is the participant's typical method of communication with his caregivers and visitors.

Typing rate assessment task design

To compare with the neural-based decoding rates achieved with our system, we measured the participant's typing rate while he used his typing interface in a custom task. In each trial of this task, we presented a word or sentence on the screen and the participant typed out that word or sentence using his typing interface. We instructed the participant to not use any of the word suggestion or completion options in his interface, but use of correction features (such as backspace or undo options) was permitted. We measured the amount of time between when the target word or sentence first appeared on the screen and when the participant entered the final letter of the target. We then used this duration and the target word or utterance to measure words per minute and correct characters per minute for each trial.

We used a total of 35 trials (25 words and 10 sentences). Punctuation was included when presented to the participant, but the participant was instructed not to type out punctuation during the task. The target words and sentences were:

1. `Thirsty`
2. `I`
3. `Tired`
4. `Are`
5. `Up`
6. `How`
7. `Outside`
8. `You`
9. `Bad`
10. `Clean`
11. `Have`
12. `Tell`

13. Hello
14. Going
15. Right
16. Closer
17. What
18. Success
19. It
20. Family
21. That
22. Help
23. Do
24. Am
25. Okay
26. It is good.
27. I am thirsty.
28. They are coming here.
29. Are you going outside?
30. I am outside.
31. Faith is good.
32. My family is here.
33. Please tell my family.
34. My glasses are comfortable.
35. They are coming outside.

Typing rate results and discussion

Across all trials of this typing task, the mean ± standard deviation of the participant's typing rate was 5.03 ± 3.24 correct words per minute or 17.9 ± 3.47 correct characters per minute.

Although these typing rates are slower than the real-time decoding rates of our approach, the unrestricted vocabulary size of the typing interface is a key advantage over our approach. Given the correct characters per minute that the participant is able to achieve with the typing interface, replacing the letters in the interface with the 50 words from this task could result in higher decoding rate and accuracy than what was achieved with our approach. However, this typing interface is less natural and appears to require more physical exertion than attempted speech, suggesting that the typing interface might be more fatiguing than our approach. More comments concerning the goals of this work as a proof-of-concept demonstration of a direct-speech neuroprosthesis are given in the Discussion section of the main text.

Section S2. Neural data acquisition and real-time processing

Initial data acquisition and preprocessing steps

The implanted electrocorticography (ECoG) array (PMT Corporation) contains electrodes arranged in a 16-by-8 lattice formation with 4-mm center-to-center spacing. The rectangular ECoG array has a length of 6.7 cm, a width of 3.5 cm, and a thickness of 0.51 mm, and the electrode contacts are disc-shaped with 2-mm contact diameters. To process and record neural data, signals were acquired from the ECoG array and processed in several steps involving multiple hardware devices (Figure S2 and Figure S3). First, a headstage (a detachable digital link; Blackrock Microsystems) connected to the percutaneous pedestal connector (Blackrock Microsystems) acquired electrical potentials from the implanted electrode array. The pedestal is a male connector and the headstage is a female connector. This headstage performed band-pass filtering on the signals using a hardware-based Butterworth filter between 0.3 Hz and 7.5 kHz. Afterwards, the digitized signals (with 16-bit, 250-nV per bit resolution) were transmitted through an HDMI cable to a digital hub (Blackrock Microsystems), which then sent the data through an optical fiber cable to a Neuroport system (Blackrock Microsystems). In early recording sessions, before the digital headstage was approved for use in human research, we used a human patient cable (Blackrock Microsystems) to connect the pedestal to a front-end amplifier (Blackrock Microsystems), which amplified and digitized the signals before they were sent through an optical fiber to the Neuroport system. This Neuroport system sampled all 128 channels of ECoG data at 30 kHz, applied software-based line noise cancellation, performed anti-aliasing low-pass filtering at 500 Hz, and then streamed the processed signals at 1 kHz to a separate real-time processing machine (Colfax International). The Neuroport system also acquired, streamed, and stored synchronized recordings of the relevant acoustics at 30 kHz (microphone input and speaker output from the real-time processing computer).

Further preprocessing and feature extraction

The real-time processing computer, which is a Linux machine (64-bit Ubuntu 18.04, 24 Intel Xeon Gold 6146 3.20 GHz processors, 500 GB of RAM), used a custom software package called real-time Neural Speech Recognition (rtNSR) [1, 2] to analyze and process the incoming neural data, run the tasks, perform real-time decoding, and store task data and metadata to disk. Using this software, we performed the following preprocessing steps on all acquired neural signals in real time:

1. We applied a common average reference to each time sample of the acquired ECoG data (across all electrodes), which is a standard technique for reducing shared noise in multi-channel data [3, 4].
2. We applied eight band-pass finite impulse response (FIR) filters with logarithmically increasing center frequencies in the high gamma band (at 72.0, 79.5, 87.8, 96.9, 107.0, 118.1, 130.4, and 144.0 Hz, rounded to the nearest decimal place). Each of these 390th-order filters was designed using the Parks-McClellan algorithm [5].
3. We computed analytic amplitude values for each band and channel using a 170th-order FIR filter designed with the Parks-McClellan algorithm to approximate the Hilbert

transform. For each band and channel, we estimated the analytic signal by using the original signal (delayed by 85 samples, which is half of the filter order) as the real component and the Hilbert transform of the original signal (approximated by this FIR filter) as the imaginary component [6]. Afterwards, we obtained analytic amplitude values by computing the magnitude of each of these analytic signals. We only applied this analytic amplitude calculation to every fifth sample of the band-passed signals, yielding analytic amplitudes decimated to 200 Hz.

4. We computed a single high gamma analytic amplitude measure for each channel by averaging the analytic amplitude values across the eight bands.
5. We z-scored the high gamma analytic amplitude values for each channel using Welford's method with a 30-second sliding window [7].

We used these high gamma analytic amplitude z-score time series (sampled at 200 Hz) in all analyses and during online decoding.

Portability and cost of the hardware infrastructure

In this work, the hardware used was fairly large but still portable, with most of the hardware components residing on a mobile rack with a length and width each at around 76 cm. We performed all data collection and online decoding tasks either in the participant's bedroom or in a small office room near the participant's residence. Although we supervised all use of the hardware throughout the clinical trial, the hardware and software setup procedures required to begin recording were straightforward; it is feasible that a caregiver could, after a few hours of training and with the appropriate regulatory approval, prepare our system for use by the participant without our direct supervision. To set up the system for use, a caregiver would perform the following steps:

1. Remove and clean the percutaneous connector cap, which protects the external electrical contacts on the percutaneous connector while the system is not being used.
2. Clean percutaneous connector, the digital link, and the scalp area around the percutaneous connector.
3. Connect the digital link to the percutaneous connector.
4. Turn on computers and start the software.
5. Ensure that the screen was properly positioned in front of the participant for use.

Afterwards, to disengage the system, a caregiver would perform the following steps:

1. Close the software and turn off the computers.
2. Disconnect the digital link from the percutaneous connector.
3. Clean percutaneous connector, the digital link, and the scalp area around the percutaneous connector.
4. Place the percutaneous connector cap back on the percutaneous connector.

The full hardware infrastructure was fairly expensive, primarily due to the relatively high cost of a new Neuroport system (compared to the costs of other hardware devices used in this work). However, recent work has demonstrated that a relatively cheap and portable brain-computer interface system can be deployed without a significant decrease in system

performance (compared to a typical system containing Blackrock Microsystem devices, such as the system used in this work) [8]. The demonstrations in that work suggest that future iterations of our hardware infrastructure could be made cheaper and more portable without sacrificing decoding performance.

Computational modeling infrastructure

We uploaded the collected data from the real-time processing computer to our lab's computational and storage server infrastructure. Here, we fit and optimized the decoding models, using multiple NVIDIA V100 GPUs to reduce computation time. Finalized models were then downloaded to the real-time processing computer for online decoding.

Section S3. Task design

All data were collected as a series of "blocks", with each block lasting about 5 or 6 minutes and consisting of multiple trials. There were two types of tasks: an isolated word task and a sentence task.

Isolated word task

In the isolated word task, the participant attempted to produce individual words from a 50-word set while we recorded his cortical activity for offline processing. This word set was chosen based on the following criteria:

- The ease with which the words could be used to create a variety of sentences.
- The ease with which the words could be used to communicate basic caregiving needs.
- The participant's interest in including the words. We iterated through a few versions of the 50-word set with the participant using feedback that he provided to us through his commercially-available assistive communication technology.
- The desire to include a number of words that is large enough to create a meaningful variety of sentences but small enough to enable satisfactory neural-based classification performance. This latter criterion was informed by exploratory, preliminary assessments with the participant following device implantation (prior to collection of any of the data analyzed in this study).

A list of the words contained in this 50-word set is provided at the end of this section.

To keep task blocks short in duration, we arbitrarily split this word set into three disjoint subsets, with two subsets containing 20 words each and the third subset containing the remaining 10 words. During each block of this task, the participant attempted to produce each word contained in one of these subsets twice, resulting in a total of either 40 or 20 attempted word productions per block (depending on the size of the word subset). In three blocks of the third, smaller subset, the participant attempted to produce the 10 words in that subset four times each (instead of the usual two).

Each trial in a block of this task started with a blank screen with a black background. After 1 second (or, in very few blocks, 1.5 seconds), one of the words in the current word subset was shown on the screen in white text, surrounded on either side by four period characters (for example, if the current word was "Hello", the text "....Hello...." would appear). For the next 2 seconds, the outer periods on either side (the first and last characters of the displayed text string) would disappear every 500 ms, visually representing a countdown. When the final period on either side of the word disappeared, the text would turn green and remain on the screen for 4 seconds. This color transition from white to green represented the go cue for each trial, and the participant was instructed to attempt to produce the word as soon as the text turned green. Afterwards, the task continued to the next trial. The word presentation order was randomized within each task block. The participant chose this countdown-style task paradigm from a set of potential paradigm options that we presented to him during a presurgical interview, claiming that he was able to use the consistent countdown timing to better align his production attempts with the go cue in each trial.

Sentence task

In the sentence task, the participant attempted to produce sentences from a 50-sentence set while his neural activity was processed and decoded into text. These sentences were composed only of words from the 50-word set. These 50 sentences were selected in a semi-random fashion from a corpus of potential sentences (see Section S5). A list of the sentences contained in this 50-sentence set is provided at the end of this section. To keep task blocks short in duration, we arbitrarily split this sentence set into five disjoint subsets, each containing 10 sentences. During each block of this task, the participant attempted to produce each sentence contained in one of these subsets once, resulting in a total of 10 attempted sentence productions per block. A video recording of a block of this task with the participant is provided in Video 2.

Each trial in a block of this task started with a blank screen divided horizontally into top and bottom halves, both with black backgrounds. After two seconds, one of the sentences in the current sentence subset was shown in the top half of the screen in white text. The participant was instructed to attempt to produce the words in the sentence as soon as the text appeared on the screen at the fastest rate that he was comfortably able to. While the target sentence was displayed to the participant, his cortical activity was processed in real time by a speech detection model. Each time an attempted word production was detected from the acquired neural signals, a set of cycling ellipses (a text string that cycled each second between one, two, and three period characters) was added to the bottom half of the screen as feedback, indicating that a speech event was detected. Word classification, language, and Viterbi decoding models were then used to decode the most likely word associated with the current detected speech event given the corresponding neural activity and the decoded information from any previous detected events within the current trial. Whenever a new word was decoded, that word replaced the associated cycling ellipses text string in the bottom half of the screen, providing further feedback to the participant. The Viterbi decoding model, which maintained the most likely word sequence in a trial given the observed neural activity, often updated its predictions for previous speech events given a new speech event, causing previously decoded words in the feedback text string to change as new information became available. After a pre-determined amount of time had elapsed since the detected onset of the most recent speech event, the sentence target text turned from white to blue, indicating that the decoding portion of the trial had ended and that the decoded sentence had been finalized for that trial. This pre-determined amount of time was either 9 or 11 seconds depending on the block type (see the following paragraph). After 3 seconds, the task continued to the next trial.

We collected two types of blocks of the sentence task: optimization blocks and testing blocks. The differences between these two types of blocks are:

1. We used the optimization blocks to perform hyperparameter optimization and the testing blocks to assess the performance of the decoding system.
2. We used intermediate (non-optimized) models when collecting the optimization blocks and finalized (optimized) models when collecting the testing blocks.
3. Although detected speech attempts and decoded word sequences were always provided to the participant as feedback during this task, during collection of optimization blocks he was instructed not to repeat a word if a speech event was missed or use the feedback to alter which words he attempted to produce. We included these instructions

to protect the integrity of the data for use with the hyperparameter optimization procedure (if the participant had altered his behavior because of imperfect speech detection, discrepancies between the prompted word sequence and the word sequence that the participant actually attempted could have hindered the optimization procedure). During testing blocks, however, we encouraged the participant to take the feedback into consideration when attempting to produce the target sentence. For example, if an attempted word production was not detected, the participant could repeat the production attempt before proceeding to the next word.

4. During optimization blocks, the pre-determined amount of time that controlled when the decoded word sequence in each trial was finalized (see the previous paragraph) was set to 9 seconds. During testing blocks, this task parameter was set to 11 seconds to give the participant extra time to incorporate the provided feedback from the decoding pipeline.

We also collected a conversational variant of the sentence task to demonstrate that the decoding approach could be used in a more open-ended setting in which the participant could generate custom responses to questions from the 50 words. In this variant of the task, instead of being prompted with a target sentence to attempt to repeat, the participant was prompted with a question or statement that mimicked a conversation partner and was instructed to attempt to produce a response to the prompt. Other than the conversational prompts and this change in task instructions to the participant, this variant of the task was identical to the regular version. We did not perform any analyses with data collected from this variant of the sentence task; it was used for demonstration purposes only. This variant of the task is shown in Figure 1 in the main text and Video 1.

Word and sentence lists

The 50-word set used in this work is:

1. `Am`
2. `Are`
3. `Bad`
4. `Bring`
5. `Clean`
6. `Closer`
7. `Comfortable`
8. `Coming`
9. `Computer`
10. `Do`
11. `Faith`
12. `Family`
13. `Feel`
14. `Glasses`
15. `Going`
16. `Good`
17. `Goodbye`

18. Have
19. Hello
20. Help
21. Here
22. Hope
23. How
24. Hungry
25. I
26. Is
27. It
28. Like
29. Music
30. My
31. Need
32. No
33. Not
34. Nurse
35. Okay
36. Outside
37. Please
38. Right
39. Success
40. Tell
41. That
42. They
43. Thirsty
44. Tired
45. Up
46. Very
47. What
48. Where
49. Yes
50. You

The 50-sentence set used in this work is:

1. Are you going outside?
2. Are you tired?
3. Bring my glasses here
4. Bring my glasses please
5. Do not feel bad
6. Do you feel comfortable?
7. Faith is good
8. Hello how are you?
9. Here is my computer
10. How do you feel?

11. How do you like my music?
12. I am going outside
13. I am not going
14. I am not hungry
15. I am not okay
16. I am okay
17. I am outside
18. I am thirsty
19. I do not feel comfortable
20. I feel very comfortable
21. I feel very hungry
22. I hope it is clean
23. I like my nurse
24. I need my glasses
25. I need you
26. It is comfortable
27. It is good
28. It is okay
29. It is right here
30. My computer is clean
31. My family is here
32. My family is outside
33. My family is very comfortable
34. My glasses are clean
35. My glasses are comfortable
36. My nurse is outside
37. My nurse is right outside
38. No
39. Please bring my glasses here
40. Please clean it
41. Please tell my family
42. That is very clean
43. They are coming here
44. They are coming outside
45. They are going outside
46. They have faith
47. What do you do?
48. Where is it?
49. Yes
50. You are not right

Section S4. Collection of the sentence corpus

To train a domain-specific language model for the sentence task (and to obtain a set of target sentences for this task), we used an Amazon Mechanical Turk task to crowdsource an unbiased corpus of natural English sentences that only contained words from the 50-word set. A web-based interface was designed to display the 50 words, and Mechanical Turk workers (referred to as "Turkers") were instructed to construct sentences that met the following criteria:

- Each sentence should only consist of words from the 50-word set.
- No duplicates should be present in the sentence responses for each individual Turker.
- Each sentence should be grammatically valid.
- Each sentence should have a length of 8 words or fewer.

Additionally, the Turkers were encouraged to use different words across the different sentences (while always restricting words to the 50-word set). Only Turkers from the USA were allowed for this task to restrict dialectal influences in the collected sentences. After removing spurious submissions and spammers, the corpus contained 3415 sentences (1207 unique sentences) from 187 Turkers.

Section S5. Creation of the sentence target set

To extract the set of 50 sentences used as targets in the sentence task from the Amazon Mechanical Turk corpus (refer to Section S4 for more details about this corpus), we first restricted this selection process to only consider sentences that appeared more than once in the corpus. We imposed this inclusion criterion to discourage the selection of idiosyncratic sentences for the target set. Afterwards, we randomly sampled from the remaining sentences, discarding some samples if they contained grammatical mistakes or undesired content (such as "Family is bad"). After a set of 50 sentence samples was created, a check was performed to ensure that at least 90% of the words in the 50-word set appeared at least once in this sentence set. If this check failed, we ran the sentence sampling process again until the check was passed, yielding the target sentence set for the sentence task.

During the sentence sampling procedure that ultimately yielded the 50-sentence set used in this study, the following 22 sentences were discarded:

1. `Good family is success`
2. `Tell success`
3. `Bring computer`
4. `Tell that family`
5. `I going outside`
6. `You are hungry`
7. `I feel very bad`
8. `I need glasses`
9. `I need computer`
10. `You need my help`
11. `You are coming closer`
12. `Tell you right`
13. `I am closer`
14. `It is bad outside`
15. `Success is not coming`
16. `I like nurse`
17. `Family is bad`
18. `I tell you`
19. `That nurse is thirsty`
20. `Need help`
21. `They are very thirsty`
22. `Where is computer`

The target sentence set contained 45 of the 50 possible words. The following 5 words did not appear in the target sentence set:

1. Closer
2. Goodbye
3. Help
4. Success
5. Up

However, because the word classifier was trained on isolated attempts to produce each word in the 50-word set and computed probabilities across all 50 words during inference, these 5 words could still appear in the sentences decoded from the participant's neural activity.

Section S6. Data organization

Isolated word data: Subset creation

In total, we collected 22 hours and 30 minutes of the isolated word task in 291 task blocks across 48 sessions (days of recording), with 196 trials (attempted productions) per word (9800 trials total). We split these blocks into 11 disjoint subsets: a single optimization subset and 10 cross-validation subsets. The optimization subset contained a total of 16 trials per word, and each cross-validation subset contained 18 trials per word.

To create subsets that were similarly distributed across time, we first ordered the blocks chronologically. Next, we assigned the blocks that occurred at evenly spaced indices within this ordered list (spanning from the earliest to the latest blocks) to the optimization subset. We then assigned the remaining blocks to the cross-validation subsets by iterating through the blocks while cycling through the cross-validation subset labels. We deviated slightly from this approach only to ensure that each subset contained the desired number of trials per word. This prevented any single subset from having an over-representation of data from a specific time period, although our irregular recording schedule prevented the subsets from containing blocks that were equally spaced in time (see Figure S4).

We evaluated models on data in the optimization subset during hyperparameter optimization (see Section S7). We used the hyperparameter values found during this process for all isolated word analyses, unless otherwise stated.

Using the hyperparameter values found during this process, we performed 10-fold cross-validation with the 10 cross-validation subsets, fitting our models on 9 of the subsets and evaluating on the held-out subset in each fold. Unless stated otherwise, the trials in the optimization subset were not used directly during isolated word evaluations.

Isolated word data: Learning curve scheme

To assess how quantity of training data affected performance, we used the 10 cross-validation subsets to generate a learning curve scheme. In this scheme, the speech detector and word classifier were assessed using cross-validation with nine different amounts of training data. Specifically, for each integer value of $N \in [1, 9]$, we performed 10-fold cross-validated evaluation with the isolated word data while only training on N randomly selected subsets in each fold. Through this approach, all of the available trials were evaluated for each value of N even though the amount of training data varied, and there was no overlap between training and testing data in any individual assessment. The final set of analyses in this learning curve scheme (with $N = 9$) was equivalent to a full 10-fold cross-validation analysis with all of the available data, and, with the exception of the learning curve results, we only used this set of analyses to compute all of the reported isolated word results (including the electrode contributions and confusion matrix shown in Figure 3 in the main text). With 18 attempted productions per word in each subset, the nine sets of analyses using this learning curve scheme contained 18, 36, 54, 72, 90, 108, 126, 144, and 162 trials per word during training, in that order. Because the word classifier was fit using curated detected events, not every trial was evaluated in each set of analyses (see Section S13 and Section S8 for more details).

Isolated word data: Stability subsets

To assess how stable the signals driving word detection and classification were throughout the study period, we used the isolated word data to define four date-range subsets containing data collected during different date ranges. These date-range subsets, named "Early", "Middle", "Late", and "Very late", contained data collected 9–18, 18–30, 33–41, and 88–90 weeks post-implantation, respectively. Data collected on the day of the exact 18-week mark was considered to be part of the "Early" subset, not the "Middle" subset. Each of these subsets contained 20 trials for each word, randomly drawn (without replacement) from the available data in the corresponding date range. Trials were only sampled from the isolated word cross-validation subsets (not from the optimization subset). In Figure 4 in the main text, the date ranges for these subsets are expressed relative to the start of data collection for this study (instead of being expressed relative to the device implantation date). Within each of these subsets, we further split the data into 10 disjoint subsets (referred to in this section as "pieces" to disambiguate these subsets from the four date-range subsets), each containing 2 trials of each word. Using these four date-range subsets, we defined three evaluation schemes: a within-subset scheme, an across-subset scheme, and a cumulative-subset scheme.

The within-subset scheme involved performing 10-fold cross-validation using the 10 pieces within each date-range subset. Specifically, each piece in a date-range subset was evaluated using models fit on all of the data from the remaining pieces of that date-range subset. We used the within-subset scheme to detect all of the speech events for the word classifier to use during training and testing (for each date-range subset and each evaluation scheme). The training data used within each individual cross-validation fold for each date-range subset always consisted of 18 trials per word.

The across-subset scheme involved evaluating the data in a date-range subset using models fit on data from other date-range subsets. In this scheme, the within-subset scheme was replicated, except that each piece in a date-range subset was evaluated using models fit on 6 trials per word randomly sampled (without replacement) from each of the other date-range subsets. The training data used within each individual cross-validation fold for each date-range subset always consisted of 18 trials per word.

The cumulative-subset scheme involved evaluating the data from the "Very late" subset using models fit with varying amounts of data. In this scheme, four cross-validated evaluations were performed (using the 10 pieces defined for each date-range subset). In the first evaluation, data from the "Very late" subset were analyzed by the word classifier using 10-fold cross-validation (this was identical to the "Very late" within-subset evaluation). In the second evaluation, the cross-validated analysis from the first evaluation was repeated, except that all of the data from the "Late" subset was added to the training dataset for each cross-validation fold. The third evaluation was similar except that all of the data from the "Middle" and "Late" subsets were also included during training, and in the fourth evaluation all of the data from the "Early", "Middle", and "Late" subsets were included during training.

Refer to Section S14 for a description of how these schemes were used to analyze signal stability.

Sentence data

In total, we collected 2 hours and 4 minutes of the sentence task in 25 task blocks across 7 sessions (days of recording), with 5 trials (attempted productions) of each sentence (250 trials total). We split these blocks into two disjoint subsets: A sentence optimization subset and a sentence testing subset. We used the sentence optimization subset, which contained 2 trials of each sentence (100 trials total), to optimize our sentence decoding pipeline prior to online testing. When collecting these blocks, we used non-optimized models. Afterwards, we used the data from these blocks to optimize our models for online testing (refer to the hyperparameter optimization procedure described in Section S7). These blocks were only used for optimization and were not included in further sentence decoding analyses.

We used the outcomes of the blocks contained in the testing subset, which contained 3 trials of each sentence (150 trials total), to evaluate decoding performance. These blocks were collected using optimized models.

We did not fit any models directly on neural data collected during the sentence task (from either subset).

Section S7. Hyperparameter optimization

To find optimal values for the model hyperparameters used during performance evaluation, we used hyperparameter optimization procedures to evaluate many possible combinations of hyperparameter values, which were sampled from custom search spaces, with objective functions that we designed to measure model performance. During each hyperparameter optimization procedure, a desired number of combinations were tested, and the combination associated with the lowest (best) objective function value across all combinations was chosen as the optimal hyperparameter value combination for that model and evaluation type. The data used to measure the associated objective function values were distinct from the data that the optimal hyperparameter values would be used to evaluate (hyperparameter values used during evaluation of a test set were never chosen by optimizing on data in that test set). We used three types of hyperparameter optimization procedures to optimize a total of 9 hyperparameters (see Table S1 for the hyperparameters and their optimal values).

Speech detection optimization with isolated word data

To optimize the speech detector with isolated word data, we used the `hyperopt` Python package [9], which samples hyperparameter value combinations probabilistically during the optimization procedure. We used this procedure to optimize the smoothing size, probability threshold, and time threshold duration hyperparameters (described in Section S8). Because these thresholding hyperparameters were only applied after speech probabilities were predicted, these hyperparameters did not affect training or evaluation of the artificial neural network model driving the speech detector. In each iteration of the optimization procedure, the current hyperparameter value combination was used to generate detected speech events from the existing speech probabilities. We used the objective function given in Equation S7 to measure the model performance with each hyperparameter value combination. In each detection hyperparameter optimization procedure, we evaluated 1000 hyperparameter value combinations before stopping.

As described in Section S6, we computed speech probabilities for isolated word blocks in each of the 10 cross-validation data subsets using a speech detection model trained on data from the other 9 cross-validation subsets. To compute speech probabilities for the blocks in the optimization subset, we used a speech detection model trained on data from all 10 of the cross-validation subsets. Afterwards, we performed hyperparameter optimization with the blocks in the optimization subset, which yielded the optimal hyperparameter value combination that was used during evaluation of the data in the 10 cross-validation subsets (including learning curve and stability analyses).

To generate detected events for blocks in the optimization subset (which were used during hyperparameter optimization of the word classifier), we performed a separate hyperparameter optimization with a subset of data from the 10 cross-validation subsets. This subset, containing 16 trials of each of the 50 words, was created by randomly selecting blocks from the 10 cross-validation subsets. We then performed hyperparameter optimization with this new subset using the predicted speech probabilities that had already been computed for those blocks (as described in the previous paragraph). Afterwards, we used the resulting optimal hyperparameter value combination to detect speech events for blocks in the optimization

subset.

Word classification optimization with isolated word data

To optimize the word classifier with isolated word data, we used the `Ray` Python package [10], which performs parallelized hyperparameter optimization with randomly sampled hyperparameter value combinations from pre-defined search spaces. This hyperparameter optimization approach uses a scheduler based on the "Asynchronous Successive Halving Algorithm" (ASHA) [11], which performs early stopping to discard underperforming hyperparameter value combinations before they are fully evaluated. This approach has been shown to outperform Bayesian hyperparameter optimization approaches when the computational complexity associated with the evaluation of a single hyperparameter value combination is high and a large number of hyperparameter combinations are evaluated [10]. We used this approach to optimize the word classification hyperparameters because of the long computation times required to train the ensemble of deep artificial neural network models comprising each word classifier. Training a single network on an NVIDIA V100 GPU required approximately 28 seconds per epoch using our augmented dataset. Each network required, on average, approximately 25 epochs of training (although the duration of each epoch can vary due to early stopping). This approximation indicates that a single network required 700 seconds to train. Because we used an ensemble of 4 networks during hyperparameter optimization, a total GPU time of approximately 46 minutes and 40 seconds was required to train a word classifier for a single hyperparameter value combination (for the word classifiers used during evaluation and real-time prediction, which each contained an ensemble of 10 networks, the approximate training time per classifier was 1 hour, 56 minutes, and 40 seconds). To evaluate a large number of hyperparameter value combinations given these training times, it was beneficial to use a computationally efficient hyperparameter optimization algorithm (such as the ASHA algorithm used here).

We performed two different hyperparameter optimizations for the word classifier, both using cross-entropy loss on a held-out set of trials as the objective function during optimization (see Equation S8). Each optimization evaluated 300 different combinations of hyperparameter values. For the first optimization, we used the optimization subset as the held-out set while training on data from all 10 cross-validation subsets. We used the resulting hyperparameter value combination for the isolated word analyses. For the second optimization, we created a held-out set by randomly selecting (without replacement) 4 trials of each word from blocks collected within three weeks of the online sentence decoding test blocks. The training set for this optimization contained all of the isolated word data (from the cross-validation and optimization subsets) except for the trials in this held-out set. We used the resulting optimal hyperparameter value combination during offline optimization of other hyperparameters related to sentence decoding and during online sentence decoding.

Optimization with sentence data

Using the sentence optimization subset, we performed hyperparameter optimization of the threshold detection hyperparameters (for the speech detector; see Section S8), the initial word smoothing value (for the language model; see Section S10), and the language model

scaling factor (for the Viterbi decoder; see Section S11). In this procedure, we first used the speech detector (trained on all isolated word data, including the isolated word optimization subset) to predict speech probabilities for all of the sentence optimization blocks. Then, using these predicted speech probabilities, the word classifier trained and optimized on the isolated word data for use during sentence decoding, and the language model and Viterbi decoder, we performed hyperparameter optimization across all optimization sentence blocks (see Section S6). We used the mean decoded word error rate across trials (computed by evaluating the detected events in each trial with the word classifier, language model, and Viterbi decoder) as the objective function during hyperparameter optimization. Using the `hyperopt` Python package [9], we evaluated 100 hyperparameter value combinations during optimization. We used the resulting optimal hyperparameter value combination during collection of the sentence testing blocks with online decoding.

Section S8. Speech detection model

Data preparation for offline training and evaluation

For supervised training and evaluation of the speech detector with the isolated word data, we assigned speech event labels to the neural time points. We used the task timing information during these blocks to determine the label for each neural time point. We used three types of speech event labels: preparation, speech, and rest.

Within each isolated word trial, the target utterance appeared on the screen with the countdown animation, and 2 seconds later the utterance turned green to indicate the go cue. We labeled all neural time points collected during this 2-second window ($[-2, 0]$ seconds relative to the go cue) as preparation. Relative to the go cue, we labeled neural time points collected between $[0.5, 2]$ seconds as speech and points collected between $[3, 4]$ as rest. To reduce the impact that variability in the participant's response times had on training, we excluded the time periods of $[0, 0.5]$ and $[2, 3]$ seconds relative to the go cue (the time periods surrounding the speech time period) from the training datasets. During evaluation, these time periods were labeled as preparation and rest, respectively.

We included the preparation label to enable the detector to neurally disambiguate attempted speech production from speech preparation. This was motivated by the assumption that neural activity related to attempted speech production would be more readily discriminable by the word classifier than activity related to speech preparation.

After labeling the neural time points, we defined 500 ms windows of neural activity from the data in each block. Each window of neural data corresponded to 500 ms of neural activity from all electrode channels. These windows were highly overlapping, shifting by only one neural sample (5 ms) between windows. These windows were created to use a type of model training (truncated backpropagation) that is described later in this section.

Speech detection model architecture and training

We used the `PyTorch 1.6.0` Python package to create and train the speech detection model [12].

The speech detection architecture was a stack of three long short-term memory (LSTM) layers with decreasing latent dimension sizes (150, 100, and 50) and a dropout of 0.5 applied at each layer. Recurrent layers are capable of maintaining an internal state through time that can be updated with new individual time samples of input data, making them well suited for real-time inference with temporally dynamic processes [13]. We use LSTMs specifically because they are better suited to model long-term dependencies compared to the original recurrent layer. The LSTMs are followed by a fully connected layer to project the last latent dimensions to probabilities across the three classes (rest, speech, and preparation). A similar model has been used to detect overt speech in a recent study [14], although our architecture was designed independently. A schematic depiction of this architecture is given in Figure S5.

Let $\mathbf{y}$ denote a series of neural data windows and $\boldsymbol{\ell}$ denote a series of corresponding labels for those windows, with y_n as the data window at index n in the data series and ℓ_n as the corresponding label at index n in the label series. The speech detection model outputs a distribution of probabilities $Q(\ell_n \mid y_n)$ over the three possible values of ℓ_n from the set of state labels $L = \{\text{rest}, \text{preparation}, \text{speech}\}$. The predicted distribution Q implicitly depends on

the model parameters. We trained the speech detection model to minimize the cross-entropy loss of this distribution with respect to the true distribution using the data and label series, represented by the following equation:

$$\begin{aligned} \mathrm{H}_{P,Q}\left(\boldsymbol{\ell} \mid \mathbf{y}\right) &= \mathbb{E}_P\left[-\log Q\left(\boldsymbol{\ell} \mid \mathbf{y}\right)\right] \\ &\approx -\frac{1}{N}\sum_{n=1}^{N}\log Q\left(\ell_n \mid y_n\right), \end{aligned} \tag{S1}$$

with the following definitions:

- P : The true distribution of the states, determined by the assigned state labels $\boldsymbol{\ell}$.
- N : The number of samples.
- $\mathrm{H}_{P,Q}\left(\boldsymbol{\ell} \mid \mathbf{y}\right)$: The cross entropy of the predicted distribution with respect to the true distribution for $\boldsymbol{\ell}$.
- log : The natural logarithm.

Here, we approximate the expectation of the true distribution with a sample average under the observed data with N samples.

During training, a false positive weighting of 0.75 was applied to any frame where the speech label was falsely predicted. With this modification, the cross-entropy loss from Equation S1 is redefined as:

$$\mathrm{H}_{P,Q}\left(\boldsymbol{\ell} \mid \mathbf{y}\right) \approx -\frac{1}{N}\sum_{n}^{N} w_{\mathrm{fp},n}\log Q(\ell_n \mid y_n), \tag{S2}$$

where $w_{\mathrm{fp},n}$ is the false positive weight for sample n and is defined as:

$$w_{\mathrm{fp},n} \coloneqq \begin{cases} 0.75 & \text{if } \left(\ell_n \neq \text{speech}\right) \text{ and } \left(\underset{l \in L}{\operatorname{argmax}}\left[Q\left(l \mid y_n\right)\right] = \text{speech}\right) \\ 1 & \text{otherwise.} \end{cases} \tag{S3}$$

As a result of this weighting, the loss associated with a sample that was incorrectly classified as occurring during a speech production attempt was only weighted 75% as much as the other samples. This weighting was only applied during training of speech detection models that were used to evaluate isolated word data. We applied this weighting to encourage the model to prefer detecting full speech events, which discouraged fluctuating speech probabilities during attempted speech productions that could prevent a production attempt from being detected. This effectively increased the number of isolated word trials that had an associated detected speech event during training and evaluation of the word classifier.

Typically, LSTM models are trained with backpropagation through time (BPTT), which unrolls the backpropagation through each time step of processing [15]. Due to the periodicity of our isolated word task structure, it is possible that relying only on BPTT would cause the model to learn this structure and predict events at every go cue instead of trying to learn neural indications of speech events. To prevent this, we used *truncated* BPTT, an approach that limits how far back in time the gradient can propagate [16, 17]. We manually

implemented this by defining 500 ms sliding windows in the training data (as described earlier). We used these windows as the y_n values during training, with ℓ_n equal to the label assigned to the final time point in the window. By processing the training data in windows, this forced the gradient to only backpropagate 500 ms at a time, which was not long enough to learn the periodicity of the task (the time between each trial's go cue was typically 7 seconds). During online and offline inference, the data was not processed in windows and was instead processed time point by time point.

During training, we used the Adam optimizer to minimize the cross entropy given in Equation S2 [18], with a learning rate of 0.001 and default values for the remaining Adam optimization parameters. When evaluating the speech detector on isolated word data, we used the 10-fold cross-validation scheme described in Section S6. When performing offline and online inference on sentence data, we used a version of the speech detector that was trained on all of the isolated word data in the 10 cross-validation subsets. During training, the training set was further split into a training set and a validation set, where the validation set was used to perform early stopping. We trained the model until model performance did not improve (if cross-entropy loss on the validation set was not lower than the lowest value plus a loss tolerance value computed in a previous epoch) for 5 epochs in a row and at least 10 epochs had been completed, at which point model training was stopped and the model parameters associated with the lowest loss were saved. The loss tolerance value was set to 0.001, although it did not seem to have significant impact on model training.

Speech event detection

During testing, the neural network predicted probabilities for each class (rest, preparation, speech) given the input neural data from a block. To detect attempted speech events, we applied thresholding to the predicted speech probabilities. This thresholding approach is identical to the approach we used in our previous work [2]. First, we smoothed the probabilities using a sliding window average. Next, we applied a threshold to the smoothed probabilities to binarize each frame (with a value of 1 for speech and 0 otherwise). Afterwards, we "debounced" these binarized values by applying a time threshold. This debouncing step required that a change in the presence or absence of speech (as indicated by the binarized values) be maintained for a minimum duration before the detector deemed it as an actual change. Specifically, a speech onset was only detected if the binarized value changed from 0 to 1 and remained 1 for a pre-determined number of time points (or longer). Similarly, a speech offset was only detected if the binarized value changed from 1 to 0 and remained 0 for the same pre-determined number of time points (or longer).

This process of obtaining speech events from the predicted probabilities was parameterized by three detection thresholding hyperparameters: the size of the smoothing window, the probability threshold value, and the time threshold duration. We used hyperparameter optimization to determine values for these parameters (described later in this section and in Section S7).

Detection score and hyperparameter optimization

During hyperparameter optimization of the detection thresholding hyperparameters with the isolated word data, we used an objective function derived from a variant of the detection score metric used in our previous work [2]. The detection score is a weighted average of frame-level and event-level accuracies for each block.

The frame-level accuracy measures the speech detector's ability to predict whether or not a neural time point occurred during speech. Ideally, the speech detector would detect events that spanned the duration of the actual attempted speech event (as opposed to detecting small subsets of each actual speech event, for example). We defined frame-level accuracy a_{frame} as:

$$a_{\text{frame}} \coloneqq \frac{w_{\text{p}} F_{\text{TP}} + (1 - w_{\text{p}}) F_{\text{TN}}}{w_{\text{p}} F_{\text{P}} + (1 - w_{\text{p}}) F_{\text{N}}}, \tag{S4}$$

with the following variable definitions:

- w_{p} : The positive weight fraction, which we used to control the importance of correctly detecting positive frames (correctly identifying which neural time points occurred during attempted speech) relative to negative frames (correctly identifying which neural time points did not occur during attempted speech).
- F_{P} : The number of actual positive frames (the number of time points that were assigned the speech label during data preparation).
- F_{TP} : The number of detected true positive frames (the number of time points that were correctly identified as occurring during an attempted speech event).
- F_{N} : The number of actual negative frames (the number of time points that were labeled as preparation or rest during data preparation).
- F_{TN} : The number of detected true negative frames (the number of time points that were correctly identified as not occurring during an attempted speech event).

In this work, we used $w_{\text{p}} = 0.75$, which encouraged the speech detector to prefer making false positive errors to making false negative errors.

The event-level accuracy measures the detector's ability to detect a speech event during an attempted word production. We defined event-level accuracy a_{event} as:

$$a_{\text{event}} \coloneqq \max\left(0, \frac{E_{\text{TP}} - E_{\text{FP}} - E_{\text{FN}}}{E_{\text{P}}}\right), \tag{S5}$$

with the following variable definitions:

- E_{TP} : The number of true positive detected events (the number of detected speech events that corresponded to an actual word production attempt).
- E_{FP} : The number of false positive detected events (the number of detected speech events that did not correspond to an actual word production attempt).
- E_{FN} : The number of false negative events (the number of actual word production attempts that were not associated with any detected event).

- E_{P} : The number of actual word production attempts (the number of trials).

We calculated event-level accuracy after curating the detected events, which involved matching each trial with a detected event (or the absence of a detected event, as described later in this section). The event-level accuracy ranges from 0 to 1, with a value of 1 indicating that there were no false positive or false negative detected events.

Using these two accuracy measures, we compute the detection score as:

$$s_{\mathrm{detection}} = w_{\mathrm{F}}\, a_{\mathrm{frame}} + (1 - w_{\mathrm{F}})\, a_{\mathrm{event}}, \tag{S6}$$

where w_{F} is the frame-level accuracy weight. Because the word classifier relied on fixed-duration time windows of neural activity relative to the detected onsets of the speech events, accurately predicting the detected offsets was less important than successfully detecting an event each time the participant attempted to produce a word. Informed by this, we set $w_{\mathrm{F}} = 0.4$ to assign more weight to the event-level accuracy than the frame-level accuracy.

During optimization of the three detection thresholding hyperparameters with the isolated word data, the primary goal was to find hyperparameter values that maximized detection score. We also included an auxiliary goal to select small values for the time threshold duration hyperparameter. We included this auxiliary goal because a large time threshold duration increases the chance of missing shorter utterances and, if the duration is large enough, adds delays to real-time speech detection. The objective function used during this hyperparameter optimization procedure, which encapsulated both of these goals, can be expressed as:

$$c_{\mathrm{hp}}\left(\Theta\right) = \left(1 - s_{\mathrm{detection}}\right)^2 + \lambda_{\mathrm{time}}\, \theta_{\mathrm{time}}, \tag{S7}$$

with the following variable definitions:

- $c_{\mathrm{hp}}\left(\Theta\right)$: The value of the objective function using the hyperparameter value combination Θ.
- λ_{time} : The penalty applied to the time threshold duration.
- θ_{time} : The time threshold duration value, which is one of the three parameters contained in Θ.

Here, we used $\lambda_{\mathrm{time}} = 0.00025$.

We only used this objective function during optimization of the detection models that were used to detect speech events for the isolated word trials. We used a different objective function when preparing detection models for use with the sentence data. See Section S7 and Table S1 for more information on the hyperparameter optimization procedures.

Detected event curation for isolated word data

After processing the neural data for an isolated word block and detecting speech events, we curated the detected events to match each one to an actual word production attempt (and to identify word production attempts that did not have a corresponding detected event and detected events that did not correspond to a word production attempt). We used this curation procedure to measure the number of false positive and false negative event detections

during calculation of the event-level accuracy (Equation S5) and to match trials to neural data during training and evaluation of the word classifier. We did not use this curation procedure with sentence data.

To curate detected events, we performed the following steps for each trial:

1. We identified all of the detected onsets that occurred in a time window spanning from -1.5 to $+3.5$ seconds (relative to the go cue). Any events with detected onsets outside of this time window were considered false positive events and were included when computing the value of E_{FP}.
2. If there was exactly one detected onset in this time window, we assigned the associated detected event to the trial.
3. Otherwise, if there were no detected onsets in this time window, we did not assign a detected event to the trial (this was considered a false negative event and was included when computing the value of E_{FN}).
4. Otherwise, there were two or more detected onsets in this time window, and we performed the following steps to process these detected events:
    a. If exactly one of these detected onsets occurred after the go cue, we assigned the detected event associated with that detected onset to the trial.
    b. Otherwise, if none of these detected onsets occurred after the go cue, we assigned the detected event associated with the latest detected onset to the trial (this was the detected event that had the detected onset closest to the go cue).
    c. Otherwise, if two or more detected onsets occurred after the go cue, we computed the length of each detected event associated with these detected onsets and assigned the longest detected event to the trial. If a tie occurred, we assigned the detected event with an onset closest to the go cue to the trial.
    d. Each of these detected events that were not assigned to the trial were considered false positive events and were included when computing the value of E_{FP}. Information about how many trials were processed using these steps (a–d) because they contained multiple candidates is provided in the caption for Figure S8.

Because false negatives cause some trials to not be associated with a detected event, the number of trials that actually get used in an analysis step may be less than the number of trials reported. For example, if we state that N trials of each word were used in an analysis step, the actual number of trials analyzed by the word classifier in that step may be less than N for one or more words depending on how many false negative detections there were.

Section S9. Word classification model

Data preparation for offline training and evaluation

During training and evaluation of the word classifier with the isolated word data, for each trial we obtained the time of the detected onset (if available; determined by the detection curation procedure described in Section S8). During evaluation with each trial, the word classifier predicted the probability of each of the 50 words being the target word that the participant was attempting to produce given the time window of high gamma activity spanning from -1 to $+3$ seconds relative to the detected onset.

To increase the number of training samples and improve robustness of the learned feature mapping to small temporal variabilities in the neural inputs, during model fitting we augmented the training dataset with additional copies of the trials by jittering the onset times, which is analogous to the well-established use of data augmentation techniques used to train neural networks for supervised image classification [19]. Specifically, for each trial, we obtained the neural time windows spanning from $(-1 + a)$ to $(3 + a)$ seconds relative to the detected onset for each $a \in \{-1, -0.667, -0.333, 0, 0.333, 0.667, 1\}$. Each of these time windows was included as a training sample and was assigned the associated target word from the trial as the label.

During offline and online training and evaluation, we downsampled the high gamma activity in each time window before passing the activity to the word classifier, which has been shown to improve speech decoding with artificial neural networks (ANNs) in our previous work [20]. We used the `decimate` function within the `SciPy` Python package to decimate the high gamma activity for each electrode by a factor of 6 (from 200 Hz to 33.3 Hz) [21]. This function applies an 8th-order Chebyshev type I anti-aliasing filter before decimating the signals. After decimation, we normalized each time sample of neural activity such that the Euclidean norm across all electrodes was equal to 1.

Word classification model architecture and training

We used the `TensorFlow 1.14` Python package to create and train the word classification model [22].

Within the word classification ANN architecture, the neural data was processed by a temporal convolution with a two-sample stride and two-sample kernel size, which further downsampled the neural activity in time while creating a higher-dimensional representation of the data. Temporal convolution is a common approach for extracting robust features from time series data [23]. This representation was then processed by a stack of two bidirectional gated recurrent unit (GRU) layers, which are often used for nonlinear classification of time series data [24]. Afterwards, a fully connected (dense) layer with a softmax activation projects the latent dimension from the final GRU layer to probability values across the 50 words. Dropout layers are used between each intermediate representation for regularization. A schematic depiction of this architecture is given in Figure S6.

Let $\mathbf{y}$ denote a series of high gamma time windows and $\mathbf{w}$ denote a series of corresponding target word labels for those windows, with y_n as the time window at index n in the data series and w_n as the corresponding label at index n in the label series. The word classifier outputs a distribution of probabilities $Q(w_n \mid y_n)$ over the 50 possible values of w_n from the

50-word set W. The predicted distribution Q implicitly depends on the model parameters. We trained the word classifier to minimize the cross-entropy loss of this distribution with respect to the true distribution using the data and label series, represented by the following equation:

$$\begin{aligned} \mathrm{H}_{P,Q}\left(\mathbf{w} \mid \mathbf{y}\right) &= \mathbb{E}_P\left[-\log Q\left(\mathbf{w} \mid \mathbf{y}\right)\right] \\ &\approx -\frac{1}{N}\sum_{n=1}^{N}\log Q\left(w_n \mid y_n\right), \end{aligned} \tag{S8}$$

with the following definitions:

- P : The true distribution of the labels, determined by the assigned word labels $\mathbf{w}$.
- N : The number of samples.
- $\mathrm{H}_{P,Q}\left(\mathbf{w} \mid \mathbf{y}\right)$: The cross entropy of the predicted distribution with respect to the true distribution for $\mathbf{w}$.
- log : The natural logarithm.

Here, we approximate the expectation of the true distribution with a sample average under the observed data with N samples.

During training, we used the Adam optimizer to minimize the cross entropy given in Equation S8 [18], with a learning rate of 0.001 and default values for the remaining Adam optimization parameters. Each training set was further split into a training set and a validation set, where the validation set was used to perform early stopping. We trained the model until model performance did not improve (if cross-entropy loss on the validation set was not lower than the lowest value computed in a previous epoch) for 5 epochs in a row, at which point model training was stopped and the model parameters associated with the lowest loss were saved. Training typically lasted between 20 and 30 epochs. When applying gradient updates to the model parameters after each epoch, if the Euclidean norm of the gradient across all of the parameter update values (before scaling these values with the learning rate) was greater than 1, then, to prevent exploding gradients, the gradient was normalized such that its Euclidean norm was equal to 1 [25].

To reduce overfitting on the training data, each word classifier contained an ensemble of 10 ANN models, each with identical architectures and hyperparameter values but with different parameter values (weights) [26]. During training, each ANN was initialized with random model parameter values and was individually fit using the same training samples, although each ANN processed the samples in a different order during stochastic gradient updates. This process yielded 10 different sets of model parameters. During evaluation, all 10 of the ensembled ANNs processed each input neural time window, and we averaged the predicted distribution $Q(w_n \mid y_n)$ for each ANN to compute the overall predicted word probabilities for each of the 50 possible values of w_n given the neural time window y_n.

We used a hyperparameter optimization procedure to select values for model parameters that were not directly learned during training. We computed two different hyperparameter value combinations: one for offline isolated word analyses and one for online sentence decoding. For faster hyperparameter searching, we used ensembles of 4 ANN models when performing hyperparameter optimization rather than the full set of 10. See Section S7 and Table S1 for more details.

Modifications for the sentence task

For online sentence decoding, we trained a modified version of the word classifier on all of the isolated word data. During hyperparameter optimization for this version of the word classifier, the held-out set contained 4 trials of each word randomly sampled from blocks collected near the end of the study period (see Section S7 for more details). After hyperparameter optimization, we then trained a word classifier with the selected hyperparameters by using this held-out set of 4 trials of each word as the validation set (used to perform early stopping) and all of the remaining isolated word data as the training set. During this training procedure, we added a single modification to the loss function used during training: We weighted each training sample by the occurrence frequency of the target word label within the corpus that was crowdsourced from Amazon Mechanical Turk and used to train the language model (see Section S4). Words that occurred more frequently were assigned more weight. We included this modification to encourage the word classifier to focus on correctly classifying neural time windows detected during attempted production of high-frequency words (such as "I"), at the potential cost of classification performance for low-frequency words (such as "glasses").

With this modification, the loss function from Equation S8 can be revised to:

$$\mathrm{H}'_{P,Q}\left(\mathbf{w} \mid \mathbf{y}\right) \approx -\frac{1}{N}\sum_{n=1}^{N} \xi\left(w_n\right) \log Q\left(w_n \mid y_n\right), \tag{S9}$$

with the following variable definitions:

1. $\mathrm{H}'_{P,Q}\left(\mathbf{w} \mid \mathbf{y}\right)$: The revised cross-entropy loss function.
2. $\xi\left(w_n\right)$: The word occurrence frequency weighting function.

The word occurrence frequency weighting function is defined as:

$$\xi\left(w_n\right) \coloneqq \frac{\kappa_{w_n}}{\bar{\xi} \sum\limits_{\omega \in W} \kappa_{\omega}} \tag{S10}$$

where κ_{w_n} is the number of times the target word label w_n occurred in the reference corpus, $\sum\limits_{\omega \in W} \kappa_{\omega}$ is the total number of words in the reference corpus, and W is the 50 word set.

We define $\bar{\xi}$ as:

$$\bar{\xi} \coloneqq \frac{1}{|W|} \sum_{\omega_i \in W} \frac{\kappa_{\omega_i}}{\sum_{\omega \in W} \kappa_{\omega}} \tag{S11}$$

where $|W|$ denotes the cardinality of the 50-word set (which is equal to 50). Therefore, $\bar{\xi}$ acts to scale each word frequency in Equation S10 so that the mean word occurrence frequency is 1, which scales the objective function such that the loss value is comparable with the loss value resulting from Equation S8.

Section S10. Language modeling

Model fitting and word-sequence probabilities

To fit a language model for use during sentence decoding, we first crowdsourced a training corpus using an Amazon Mechanical Turk task (see Section S4 for more details). This corpus contained 3415 sentences comprised only of words from the 50-word set. To discourage overfitting of the language model on the most common sentences, we only included a maximum of 15 instances of each unique sentence when creating the training corpus for the language model from these responses.

Next, we extracted all n-grams with $n \in \{1, 2, 3, 4, 5\}$ from each sentence in the training corpus. Here, an n-gram is a word sequence with a length of n words [27]. For example, the n-grams (represented as tuples) extracted from the sentence "I hope my family is coming" in this approach would be:

1. `(I)`
2. `(Hope)`
3. `(My)`
4. `(Family)`
5. `(Is)`
6. `(Coming)`
7. `(I, Hope)`
8. `(Hope, My)`
9. `(My, Family)`
10. `(Family, Is)`
11. `(Is, Coming)`
12. `(I, Hope, My)`
13. `(Hope, My, Family)`
14. `(My, Family, Is)`
15. `(Family, Is, Coming)`
16. `(I, Hope, My, Family)`
17. `(Hope, My, Family, Is)`
18. `(My, Family, Is, Coming)`
19. `(I, Hope, My, Family, Is)`
20. `(Hope, My, Family, Is, Coming)`

We used the n-grams extracted in this manner from all of the sentences in the training corpus to fit a 5th-order interpolated Kneser-Ney n-gram language model with the `nltk` Python package [28, 29]. A discount factor of 0.1 was used for this model, which was the default value specified within `nltk`. The details of this language model architecture, along with characterizations of its ability to outperform simpler n-gram architectures on various corpus modeling tasks, can be found in existing literature [27, 28].

Using the occurrence frequencies of specific word sequences in the training corpus (as specified by the extracted n-grams), the language model was trained to yield the conditional probability of any word occurring given the context of that word, which is the sequence of $(n-1)$ or fewer words that precede it. These probabilities can be expressed as $p\left(w_i \mid \mathbf{c}_{i,n}\right)$, where w_i is the word at position i in some word sequence, $\mathbf{c}_{i,n}$ is the context of that word

assuming it is part of an n-gram (this n-gram is a word sequence containing n words, with w_i as the last word in that sequence), and $n \in \{1, 2, 3, 4, 5\}$. The context of a word w_i is defined as the following tuple:

$$\mathbf{c}_{i,n} \coloneqq \left(w_{i-(n-1)}, \ldots, w_{i-1}\right). \tag{S12}$$

When $n = 1$, the context is (), an empty tuple. When $n = 2$, the context of a word w_i is (w_{i-1}), a single-element tuple containing the word preceding w_i. With the language model used in this work, this pattern continues up to $n = 5$, where the context of a word w_i is $(w_{i-4}, w_{i-3}, w_{i-2}, w_{i-1})$, a tuple containing the four words in the sequence that precede w_i (in order). It was required that each $w_i \in W$, where W is the 50-word set. This requirement also applied to the words contained in the contexts $\mathbf{c}_{i,n}$.

Sentence independence

During the sentence task, each sentence was decoded independently of the other sentences in the task block. The contexts $\mathbf{c}_{i,n}$ that we used during inference with the language model could only contain words that preceded, but were also in the same sentence as, w_i (contexts never spanned two or more sentences). The relationship between the values i and n in the contexts we used during inference can be expressed as:

$$n = \min\left(i + 1, m\right), \tag{S13}$$

where m is the order of the model (for this model, $m = 5$) and $i = 0$ specifies the index of the initial word in the sentence. Substituting this definition of n into the definition for $\mathbf{c}_{i,n}$ specified in Equation S12 yields:

$$\mathbf{c}_i \coloneqq \left(w_{i-\min(i,m-1)}, \ldots, w_{i-1}\right) \tag{S14}$$

where $\mathbf{c}_i$ is the context of word w_i within a sentence trial. This substitution simplifies the form of the word probabilities obtained from the language model to $p\left(w_i \mid \mathbf{c}_i\right)$.

Initial word probabilities

Because sentences were always decoded independently in this task, an empty tuple was only used as context when performing inference for w_0, the initial word in a sentence. Instead of using the values for $p\left(w_0 \mid \mathbf{c}_0\right)$ yielded by the language model during inference, we instead used word counts directly from the corpus and two different types of smoothing. First, we computed the following probabilities:

$$\phi\left(w_0 \mid \mathbf{c}_0\right) = \frac{k_{w_0} + \delta}{N + \delta\left|W\right|}, \tag{S15}$$

where k_{w_0} is the number of times that the word w_0 appeared as the initial word in a sentence in the training corpus, N is the total number of sentences in the training corpus, and δ is an additive smoothing factor. Here, the additive smoothing factor is a value that is added to all of the counts k_{w_0} prior to normalization, which smooths (reduces the variance of) the probability distribution [27]. In this work, $N = 3415$, $\delta = 3$, and $|W| = 50$.

We then smoothed these $\phi(w_0 \mid \mathbf{c}_0)$ values to further control how flat the probability distribution over the initial word probabilities was. This can be interpreted as control over how "confident" the initial word probability predictions by the language model were (flatter probability distributions indicate less confidence). We used a hyperparameter to control the extent of this smoothing, allowing the hyperparameter optimization procedure to determine how much smoothing was optimal during testing (see Section S7 and Table S1 for a description of the hyperparameter optimization procedure). We used the following equation to perform this smoothing:

$$p(w_0 \mid \mathbf{c}_0) = \frac{\phi(w_0 \mid \mathbf{c}_0)^{\psi}}{\sum_{w_j \in W} \phi(w_j \mid \mathbf{c}_0)^{\psi}}, \tag{S16}$$

where ψ is the initial word smoothing hyperparameter value. When $\psi > 1$, the variance of the initial word probabilities is increased, making them less smooth. When $\psi < 1$, the variance of the initial word probabilities is decreased, making them more smooth. When $\psi = 1$, $p(w_0 \mid \mathbf{c}_0) = \phi(w_0 \mid \mathbf{c}_0)$. Note that the denominator in Equation S16 is used to re-normalize the smoothed probabilities so that they sum to 1.

The Viterbi decoding model used in this work contained a language model scaling factor (LMSF), which is a separate hyperparameter that re-scaled the $p(w_i \mid \mathbf{c}_i)$ values during the sentence decoding approach (see Section S11 for more details). The effect that this hyperparameter had on all of the language model probabilities resembles the effect that ψ had on the initial word probabilities. This should have encouraged the hyperparameter optimization procedure to find an LMSF value that optimally scaled the language model probabilities and a value for ψ that optimally smoothed the initial word probabilities relative to the scaling that was subsequently applied to them.

Real-time implementation

To ensure rapid inference during real-time decoding, we pre-computed the $p(w_i \mid \mathbf{c}_i)$ values with the language model and smoothing hyperparameter values for every possible combination of w_i and $\mathbf{c}_i$ and then stored these values in an `hdf5` file [30]. This file served as a lookup table during real-time decoding; the values were stored in multi-dimensional arrays within the file, and efficient lookup queries to the table were fulfilled during real-time decoding using the `h5py` Python package [31]. In future iterations of this decoding approach requiring larger vocabulary sizes, it may be more suitable to use a more sophisticated language model that is also computationally efficient enough for real-time inference, such as the `kenlm` language model [32].

Section S11. Viterbi decoding

Hidden Markov model representation of the sentence decoding procedure

During a sentence trial, the relationship between the sequence of words that the participant attempted to produce and the sequence of neural activity time windows provided by the speech detector can be represented as a hidden Markov model (HMM). In this HMM, each observed state y_i is the time window of neural activity at index i within the sequence of detected time windows for any particular trial, and each hidden state q_i is the n-gram containing the words that the participant had attempted to produce from the first word to the word at index i in the sequence (Figure S7). Here, $q_i = \{w_i, \mathbf{c}_i\}$, where w_i is the word at index i in the sequence and $\mathbf{c}_i$ is the context of that word (defined in Equation S14; see Section S10).

The emission probabilities for this HMM are $p(y_i \mid q_i)$, which specify the likelihood of observing the neural time window y_i given the n-gram q_i. With the assumption that the time window of neural activity associated with the attempted production of w_i is conditionally independent of all of the other attempted word productions given w_i ($y_i \perp\!\!\!\perp w_j \mid w_i \; \forall \; j \neq i$), $p(y_i \mid q_i)$ simplifies to $p(y_i \mid w_i)$. The word classifier provided the probabilities $p(w_i \mid y_i)$, which was used directly as the values for $p(y_i \mid w_i)$ by applying Bayes' theorem and assuming a flat prior probability distribution.

The transition probabilities for this HMM are $p(q_i \mid q_{i-1})$, which specify the probability that q_i is the n-gram at index i (the sequence of at most n words, containing w_i as the final word, that the participant attempted to produce) given that the n-gram at index $(i-1)$ was q_{i-1}. Here, q_{-1} can be defined as an empty set, indicating that q_0 is the first word in the sequence. Because any elements in $\mathbf{c}_i$ will be contained in q_{i-1} and w_i is the only word in q_i that is not contained in q_{i-1}, $p(q_i \mid q_{i-1})$ simplifies to $p(w_i \mid \mathbf{c}_i)$, which were the word sequence prior probabilities provided by the language model. Implicit in this simplification is the assertion that $p(q_i \mid q_{i-1}) = 0$ if q_i is incompatible with q_{i-1} (for example, if the final word in $\mathbf{c}_i$ is not equal to the second-to-last word in q_{i-1}).

Viterbi decoding implementation

To predict the words that the participant attempted to produce during the sentence task, we implemented a Viterbi decoding algorithm with this underlying HMM structure. The Viterbi decoding algorithm uses dynamic programming to compute the most likely sequence of hidden states given hidden-state prior transition probabilities and observed-state emission likelihoods [33, 34]. To determine the most likely hidden-state sequence, this algorithm iteratively computes the probabilities of various "paths" through the hidden-state sequence space (various combinations of q_i values). Here, each of these Viterbi paths was parameterized by a particular path through the hidden states (a particular word sequence) and the probability associated with that path given the neural activity. Each time a new word production attempt was detected, this algorithm created a set of new Viterbi paths by computing, for each existing Viterbi path, the probability of transitioning to each valid new word given the detected time window of neural activity and the preceding words in the associated existing Viterbi path. The creation of new Viterbi paths from existing paths can be expressed using the following

recursive formula:

$$V_i = \left\{ v_{i-1} + \log p\left(y_i \mid q_{i,v_{i-1}}\right) + L \log p\left(q_{i,v_{i-1}} \mid q_{i-1,v_{i-1}}\right) \;\middle|\; v_{i-1} \in V_{i-1} \;\wedge\; w_i \in W \right\}, \quad \text{(S17)}$$

with the following variable definitions:

- V_j : The set of all Viterbi paths created after the word production attempt at index j within a sentence trial.
- v_j : A Viterbi path within V_j. Each of these Viterbi paths was parameterized by the n-grams $(q_0, \ldots, q_j)$ (or, equivalently, the words $(w_0, \ldots, w_j)$) and the log probability of that sequence of words occurring given the neural activity, although these equations only describe the recursive computation of the log probability values (the tracking of the words associated with each Viterbi path is implicitly assumed).
- q_{j,v_k} : The n-gram q_j, containing the word w_j and the context of that word. This context is determined from the most recent words within the hidden state sequence of Viterbi path v_k.
- $p\left(y_j \mid q_{j,v_{j-1}}\right)$: The emission probability specifying the likelihood of the observed neural activity y_j given the n-gram $q_{j,v_{j-1}}$.
- $p\left(q_{i,v_{i-1}} \mid q_{i-1,v_{i-1}}\right)$: The transition probability specifying the prior probability of transitioning to the n-gram $q_{i,v_{i-1}}$ from the n-gram $q_{i-1,v_{i-1}}$.
- L : The language model scaling factor, which is a hyperparameter that we used to control the weight of the transition probabilities from the language model relative to the emission probabilities from the word classifier (see Section S7 and Table S1 for a description of the hyperparameter optimization procedure).
- W : The 50-word set.
- log : The natural logarithm.

Using the simplifications described in the previous section, Equation S17 can be simplified to the following equation:

$$V_i = \left\{ v_{i-1} + \log p\left(w_i \mid y_i\right) + L \log p\left(w_i \mid \mathbf{c}_{i,v_{i-1}}\right) \;\middle|\; v_{i-1} \in V_{i-1} \;\wedge\; w_i \in W \right\}, \quad \text{(S18)}$$

where $\mathbf{c}_{j,v_k}$ is the context of word w_j determined from the Viterbi path v_k, $p\left(w_i \mid y_i\right)$ are the emission probabilities (obtained from the word classifier), and $p\left(w_i \mid \mathbf{c}_{i,v_{i-1}}\right)$ are the transition probabilities (obtained from the language model). At the start of each sentence trial, the index i was reset to zero (the first word in each trial was denoted w_0), and any existing Viterbi paths from a previous trial were discarded. To initialize the recursion, we defined V_{-1} as a singleton set containing a single Viterbi path with the empty set as its hidden state sequence and an associated log probability of zero. We used log probabilities in practice for numerical stability and computational efficiency.

Viterbi path pruning via beam search

As specified in Equation S18, when new emission probabilities $p\left(w_i \mid y_i\right)$ were obtained from the word classifier, our Viterbi decoder computed the new set of Viterbi paths V_i, comprised of the paths created by transitioning each existing path within V_{i-1} to each possible next n-gram q_i. As a result, the number of new Viterbi paths created for index i was equal to $|V_{i-1} \times W|$ (the number of existing Viterbi paths at index $i-1$ multiplied by 50). Without intervention, the number of Viterbi paths grows exponentially as the index increases ($|V_i| = |W|^{(i+1)}$).

To prevent exponential growth, we applied a beam search with a beam width of β to each new Viterbi path set V_i immediately after it was created. This beam search enforced a maximum size of β for each new Viterbi path set, retaining the β most likely paths (the paths with the greatest associated log probabilities) and pruning (discarding) the rest. All paths were retained if $|V_i| \leq \beta$. Expanding Equation S18 to include the beam search procedure yields the final set of Viterbi decoding update equations that we used in practice during sentence decoding:

$$V_i' = \left\{ v_{i-1} + \log p\left(w_i \mid y_i\right) + L \log p\left(w_i \mid \mathbf{c}_{i,v_{i-1}}\right) \;\middle|\; v_{i-1} \in V_{i-1} \;\wedge\; w_i \in W \right\} \tag{S19}$$

$$V_i = \left\{ \nu_{i,j} \;\middle|\; j \in \left\{0, \ldots, \min\left(\beta, |V_i'|\right) - 1\right\} \right\}, \tag{S20}$$

where V_i' is the set of all Viterbi paths created after the word production attempt at index i within a sentence trial (before pruning) and $\nu_{i,j}$ is the element at index j of a vector created by sorting the Viterbi paths in V_i' in order of descending log probability (ties are broken arbitrarily during sorting).

Section S12. Sentence decoding evaluations

We evaluated the performance of our decoding pipeline (speech detector, word classifier, language model, and Viterbi decoder) using the online predictions made during the sentence task blocks (in the testing subset; see Section S6). Specifically, we analyzed the sentences decoded in real time from the participant's neural activity during the active phase of each trial (the portion of each trial during which the participant was instructed to attempt to produce the prompted sentence target). Offline, we counted the number of false positive speech events that were erroneously detected during inactive task phases (which were ignored during real-time decoding). These false positive events only occurred during the inactive task phase at the start of a block (prior to the first trial in the block), and this count is reported in the Results section of the main text.

Word error rates and edit distances

To measure the quality of the decoding results, we computed the word error rates (WERs) between the target and decoded sentences in each trial. WER is a commonly used metric to measure the quality of predicted word sequences, computed by calculating the edit (Levenshtein) distance between a reference (target) and decoded sentence and then dividing the edit distance by the number of words in the reference sentence. Here, the edit distance measurement can be interpreted as the number of word errors in the decoded sentence (in Figure 2 in the main text, the edit distance is referred to as the "number of word errors" or the "error count"). It is computed as the minimum number of insertions, deletions, and substitutions required to transform the decoded sentence into the reference sentence. Below we demonstrate each type of edit operation that can be used to transform an example decoded sentence (on the left side of each arrow) into the target sentence "I am good". In each case, the example decoded sentence has an edit distance of 1 to the target sentence.

- Insertion: `I good` → `I am good`
- Deletion: `I am very good` → `I am good`
- Substitution: `I am going` → `I am good`

Lower edit distances and WERs indicate better performance. We computed edit distances and WERs using predictions made with and without the language model and Viterbi decoder.

To compute block-level WERs, which are shown in Figure 2A in the main text, we first computed the edit distance for each sentence trial (which are shown in Figure 2D in the main text). We then computed the block-level WER as the sum of the edit distances across all of the trials in a test block divided by the sum of the target-sentence word lengths across all trials. This approach to measure block-level WER was preferred to simply averaging trial-level WER values because it does not overvalue short sentences compared to long ones. For example, if we simply averaged trial-level WERs to compute a block-level WER, then one error in a trial with the target sentence "I am thirsty" would cause a greater impact on WER than one error in a trial with the target sentence "My family is very comfortable", which was not a desired aspect of our block-level WER measurement.

To assess chance performance of our decoding approach with the sentence task, we measured WER using randomly generated sentences from the language model and Viterbi

decoder (independent of any neural data). To generate these sentences, we performed the following steps for each trial:

1. Start with an empty word sequence.
2. Acquire the word probabilities from the language model using the current word sequence as context.
3. Randomly sample a word from the 50-word set, using the word probabilities in step 2 as weights for the sampling.
4. Add the word from step 3 to the current word sequence.
5. Repeat steps 2–4 until the length of the current word sequence is equal to the length of the target sentence for the trial.

With the randomly generated sentence for each trial, we measured chance performance by computing block-level WERs using the method described in the preceding paragraph. This method of measuring chance performance overestimates the true chance performance because it uses the language model and the same sentence length as the target sentence for each trial (which is equivalent to assuming that the speech detection model always detected the correct number of words in each trial).

Words per minute and decoded word correctness

To measure decoding rate, we used the words per minute (WPM) metric. For each trial, we computed a WPM value by counting the number of detected words in the trial and dividing that count by the detected trial duration. We calculated each detected trial duration as the elapsed time between the time at which the sentence prompt appeared on the participant's monitor (the go cue) and the time of the last neural time sample passed from the speech detector to the word classifier in the trial.

To measure the rate at which words were accurately decoded, we also computed WPMs while only counting correctly decoded words. To determine which words were correctly decoded in each trial, we performed the following steps:

1. Start with $n = 1$ and $w = 0$.
2. Compute the WER between the first n words in the decoded sentence and the first n words in the target sentence.
3. If this WER is less than or equal to w, and if $w \neq 1$, the word at index n in the decoded sentence is deemed correct (with $n = 1$ being the index of the first word in the sentence). Otherwise, the word at index n is deemed incorrect.
4. Let w equal this WER value and increment n by 1.
5. Repeat steps 2–4 until each word in the decoded sentence has been deemed correct or incorrect.

System latency calculation

To estimate the latency of the decoding pipeline during real-time sentence decoding, we first randomly selected one of the sentence testing blocks to use to compute latencies. Because the infrastructure and model parameters were identical across sentence testing blocks, we made

the assumption that the distribution of latencies from any block should be representative of the distribution of latencies across all blocks. This was further supported by no noticeable differences in latencies across all of the sentence testing blocks (from our perspective and as attested by the participant). After randomly selecting a sentence testing block, we used a video recording of the block to identify the time at which each decoded word appeared on the screen. We then computed the latency of each real-time word prediction as the difference between the word appearance time (from the video) and the time of the final neural data point contained in the detected window of neural activity associated with the word (the final time point of neural data used by the word classifier to predict probabilities for that word production attempt, obtained from the result file associated with the block). By using these differences, the computed latencies represented the amount of time the system required to predict the next word in the sequence after obtaining all of the associated neural data that would be required to make that prediction. The timing between the video and the result file timestamps were synchronized using a short beep that is played at the start of every block (speaker output was also acquired and stored in the result file during each block; see Section S2). Across all trials, there were 42 decoded words in this block.

Using this approach, we found that the mean latency associated with the real-time word predictions was 4.0 s (with a standard deviation of 0.91 s).

Section S13. Isolated word evaluations

Classification accuracy, cross entropy, and detection errors

During offline cross-validated evaluation of the isolated word data (see Section S6), we used the word classifier to predict word probabilities from the neural data associated with the word production attempt in each trial. We computed these word probabilities using time windows of neural activity associated with curated detected events from the speech detector (see Section S8). From these predicted word probabilities, we computed classification accuracy as the fraction of trials in which the word with the highest predicted probability was the target word. In addition, we computed how often the target word was assigned one of the top 5 highest predicted probabilities by the word classifier. We also used these predicted probabilities to compute cross entropy, which measures the amount of additional information that would be required to determine the target word identities from the predicted probabilities. To compute cross entropy, we first obtained the predicted probability of the target word in each trial. The cross entropy (in bits) was then calculated as the mean of the negative log (base 2) across all of these probabilities. In addition to using the curated detected events to compute these metrics, we also used them to measure the number of detection errors made. Specifically, we measured two types of detection errors: the number of false negatives (the number of trials that were not associated with a detected event) and the number of false positives (the number of detected events were not associated with a trial). We reported these detection errors separately (classification accuracy and cross entropy were only computed with correctly detected trials and were not penalized for detection errors).

We performed these analyses using a learning curve scheme that varied the amount of data used to fit both the speech detector and word classifier (detailed in Section S6). The final set of analyses in this learning curve scheme was equivalent to using all of the available data. For every set of analyses in the learning curve scheme, the speech detector provided curated detected speech events. We used neural data aligned to the onsets of these curated detected events to fit the word classifier and predict word probabilities.

Measuring training data quantities for the learning curve scheme

Because the speech detection and word classification models used different training procedures, we measured the amount of neural data used by each type of model separately for each set of analyses in the learning curve scheme. For each word classifier, we multiplied the number of detected events used to fit the model by 4 seconds (the size of the neural time window used by the classifier). Because each set of analyses in the learning curve scheme used 10-fold cross-validation, this resulted in 10 measures of the amount of training data used for each set of analyses. By computing the mean across the 10 folds, we obtained a single measure of the average amount of data used to fit the word classifier for each set of analyses.

Each speech detection model was fit with sliding windows to predict individual time points of neural activity, resulting in many more training samples per task block than trials. Here, each training sample was a single window from the sliding window training procedure, which corresponded to an individual time point in the task block. Because we used early stopping to prevent overfitting, in practice each speech detector never used all of the data available during model fitting. However, increasing the amount of data available can increase

the diversity of the training data (for example, by having data from blocks that were collected across long time periods), which can also affect the number of epochs that the detector is trained for and the robustness of the trained detection model. To measure the amount of data available to each speech detector during training, we simply divided the number of available training samples by the sampling rate (200 Hz). To measure the amount of data that was actually used by each speech detector during training, we divided the number of training samples used by the sampling rate. By computing the mean across the 10 folds, we measured the average amount of data available and the average amount that was actually used to fit the speech detector for each set of analyses.

Electrode contributions (saliences)

To measure how much each electrode contributed to detection and classification performance, we computed electrode contributions (saliences) with the artificial neural networks (ANNs) driving the speech detection and word classification models, respectively. We used a salience calculation method that has been demonstrated with convolutional ANNs during identification of image regions that were most useful for image classification [35]. We have also used this method in our previous work to measure which electrodes were most useful for speech decoding with a recurrent and convolutional ANN [20].

To compute electrode saliences for each type of ANN, we first calculated the gradient of the loss function for the ANN with respect to the input features. The input features were individual time samples of high gamma activity across entire blocks for the speech detector or across detected time windows for the word classifier. For each input feature, we backpropagated the gradient through the ANN to the input layer. We then computed the Euclidean norm across time (within each block or trial) of the resulting gradient values associated with each electrode. Here, we used the norm of the gradient to measure the magnitude of the sensitivity of the loss function to each input (disregarding the direction of the sensitivity). Next, we computed the mean across blocks or trials of the Euclidean norm values, yielding a single salience value for each electrode. Finally, we normalized each set of electrode saliences so that they summed to 1.

We computed these saliences during the final set of analyses in the learning curve scheme, using 10-fold cross-validated evaluation of the speech detector and word classifier. We used the blocks and trials that were evaluated in the test set of each fold to compute the gradients. We also computed saliences during the signal stability analyses (see Section S14).

Information transfer rate

The information transfer rate (ITR) metric, which measures the amount of information that a system communicates per unit time, is commonly used to evaluate brain-computer interfaces [36]. Similar to formulations described in existing literature [2, 36, 37], we used the following formula to compute ITRs in this work:

$$\mathrm{ITR} = \frac{1}{T}\left[\log_2 N + P\log_2 P + (1-P)\log_2\left(\frac{1-P}{N-1}\right)\right], \tag{S21}$$

where N is the number of unique targets, P is the prediction accuracy, and T is the average time duration for each prediction. In this work, $N = 50$ (the size of the word set) and

$T = 4$ seconds (the size of the neural time window that the classifier uses to compute word probabilities). We set P equal to the mean classification accuracy for the full cross-validation analysis with the isolated word data (from the final set of analyses in the learning curve scheme). This formula makes the following assumptions:

1. On average, all possible word targets had the same prior probability (that is, the probability independent of the neural data) of being the actual word target in any trial. This is reasonable because there was an equal number of isolated word trials collected for each word target.
2. The classification accuracy used for P was representative of the overall accuracy of the word classifier (given the amount of training data) and is consistent across trials. This should be a valid assumption because our cross-validated analysis enabled us to evaluate performance across all collected trials.
3. On average, each incorrect word target had the same probability of being assigned the highest probability value in any trial. Although this is not exactly true in practice for our results (as is evident by the confusion matrix shown in Figure 3 in the main text, which shows that some words are predicted slightly more often than others on average), it is typically not exactly true in other studies that have used this formula, and it is generally regarded as an acceptable simplifying assumption.

Using Equation S21, we computed the ITR and reported the result in the caption for Figure S8.

The ITR was only computed for the isolated word predictions from the word classifier (which used the detected neural windows from the speech detector). Calculation of the ITR of the full decoding pipeline (including the language model) on sentence data would be significantly more complicated because the word-sequence probabilities from the language model will violate assumptions (1) and (3) from the list provided above [38]. The fact that some decoded sentences differed in word length from the corresponding target sentence also makes ITR computation more difficult. For simplicity, we decided to only report ITR using the word classifier outputs. This ITR measurement can also be more easily compared to the performance of the discrimination models reported in other brain-computer interface applications (independent of our specific language-modeling approach).

Investigating potential acoustic contamination

In recent work, Roussel and colleagues have demonstrated that acoustic signals can directly "contaminate" electrophysiological recordings, causing the spectrotemporal content of signals recorded via an electrophysiological recording methodology to strongly correlate with simultaneously occurring acoustic waveforms [39]. To assess whether or not acoustic contamination was present in our neural recordings, we applied the contamination identification methods described in [39] to our dataset (with some minor procedural deviations, which are noted below).

First, we randomly selected a set of 24 isolated word task blocks (which were chronologically distributed across the 81-week study period) to consider in this analysis. From each block, we obtained the neural activity recorded at 1 kHz (which was not processed using re-referencing against a common average or high gamma feature extraction) and the microphone signals

recorded at 30 kHz. These microphone signals were already synchronized to the neural signals (as described in Section S2). We then downsampled the microphone signals to 1000 Hz to match the neural data. Next, as was performed in [39], we "centered" the microphone signal by subtracting from the signal at each time point its mean value over the preceding one second.

We then computed spectrograms for the neural activity recorded from each electrode channel and the recorded microphone signal. We computed the spectrograms as the absolute value of the short-time Fourier transform. For computational efficiency, we slightly departed from [39] to use powers of two in our approach. We computed the Fourier transform within sliding windows of 256 samples (with each window containing 256 ms of data), as opposed to the 200 ms windows used in [39], resulting in 129 frequency bands with evenly spaced center frequencies between 0 and 500 Hz. Each sliding window was spaced 32 time samples apart, yielding spectrogram samples at approximately 31 Hz, as opposed to the 50 Hz rate used in [39]. Because inclusion of a large amount of "silent" task segments (segments during which the participant was not attempting to speak) would bias the analysis against finding acoustic contamination, we clipped periods of time corresponding to inter-trial silence out of the spectrograms. Specifically, we only retained the spectrograms computed from data that occurred between 0.5 seconds before and 3.5 seconds after the go cue in each trial. Although these time periods still contained samples recorded while the participant was silent, this approach drastically reduced the overall proportion of silence in the considered data.

We then measured the across-time correlations (within individual frequency bands) between each microphone spectrogram and the corresponding spectrograms for each electrode. Small correlations between a neural channel and the microphone signal is not definitive evidence of acoustic contamination; there are many factors that could influence correlation, including the presence of shared electrical noise and the characteristics of purely physiological neural responses evoked during attempted speech production. By computing correlations within narrow frequency bands, the resulting correlations are more likely (but not guaranteed) to be indicative of acoustic contamination; for example, spectral power at 300 Hz in the acoustic signal would not be expected to correlate strongly with neural oscillations at that frequency in electrophysiological signals. We aggregated the correlation matrices across spectrograms to obtain an overall correlation matrix across all the considered data, which contained one element for each electrode and frequency band. This procedure was equivalent to concatenating together the (clipped) neural and acoustic spectrograms from each block and then computing a single correlation matrix across all of the data.

To further characterize any potential acoustic contamination, we compared the correlations between the neural and acoustic spectrograms as a function of frequency against the power spectral density (PSD) of the microphone. We expected correlations to be non-zero because a core hypotheses in this work is that the neural activity recorded from the implanted electrodes is causally related to attempted speech production. However, strong correlations between the neural and acoustic spectrograms that also increase and decrease with this PSD would be strong evidence of acoustic contamination. Here, we computed the microphone PSD as the mean of the microphone spectrogram (along the frequency dimension) across all spectrogram samples and blocks (yielding a single value per frequency band).

Section S14. Stability evaluations

To assess the stability of the neural signals recorded during word production attempts, we computed classification accuracies and electrode contributions (saliences) with the speech detector and word classifier while varying the date ranges from which the data used to train and test the models were sampled. We performed these analyses using the four date-range subsets ("Early", "Middle", "Late", and "Very late") and the three evaluation schemes (within-subset, across-subset, and cumulative-subset) defined in Section S6.

First, to yield curated detected times for each subset, the speech detection model used the within-subset training scheme. As a result, all curated detected events for a subset were obtained from a speech detection model fit only with data from the same subset. The percent of trials excluded from further analysis in each subset because they were not associated with a detected event during the detected event curation procedure was 2.3%, 3.8%, 0.8%, and 1.5% for the "Early", "Middle", "Late", and "Very late" subsets, respectively. The word classifier was trained and tested using neural data aligned to the onsets of these curated detected events.

To determine if the neural signals recorded during each date range contained similar amounts of discriminatory information (and to assess the likelihood of a degradation in overall recording quality over time), we compared the classification accuracies from different date-range subsets computed using the within-subset evaluation scheme. To assess the stability of the spatial maps learned by the classification models, we also computed electrode saliences (contributions) for each date-range subset using the within-subset evaluation scheme.

To determine if the temporal proximity of training and testing data affected classification performance (and assess whether or not there were significant changes in the underlying neural activity between date-range subsets even if all of the within-subset accuracies were similar), we compared the within-subset and across-subset classification accuracies individually for each subset. The within-subset and across-subset comparisons are shown in Figure S10.

To assess whether cortical activity collected across months of recording could be accumulated to improve model performance without frequent recalibration, we computed classification accuracies on the "Very late" subset while varying the amount of training data using the cumulative-subset evaluation scheme (shown in Figure 4 in the main text). To measure training data quantities for this evaluation scheme, we used the same method as the one described in Section S13 to measure training data quantities for the word classifier in the learning curve analyses.

Section S15. Statistical testing

Word error rate confidence intervals

To compute 95% confidence intervals for the word error rates (WERs), we performed the following steps for each set of results (chance, without language model, and with language model):

1. Compile the block-level WERs into a single array (with 15 elements, one for each block).
2. Randomly sample (with replacement) 15 WER values from this array and then compute and store the median WER from these values.
3. Repeat step 2 until one million median WER values have been computed.
4. Compute the confidence interval as the 2.5 and 97.5 percentiles of the collection of median WER values from step 3.

Classification accuracy confidence intervals

To compute 95% confidence intervals for the classification accuracies obtained during the signal stability analyses, we performed the following steps for each date-range subset ("Early", "Middle", "Late", and "Very late") and each evaluation scheme (within-subset, across-subset, and cumulative-subset):

1. Compile the classification accuracies from each cross-validation fold into a single array (with 10 elements, one for each fold).
2. Randomly sample (with replacement) 10 classification accuracies from this array and then compute and store the mean classification accuracy from these values.
3. Repeat step 2 until one million mean classification accuracies have been computed.
4. Compute the confidence interval as the 2.5 and 97.5 percentiles of the collection of mean classification accuracies from step 3.

Supplementary figures

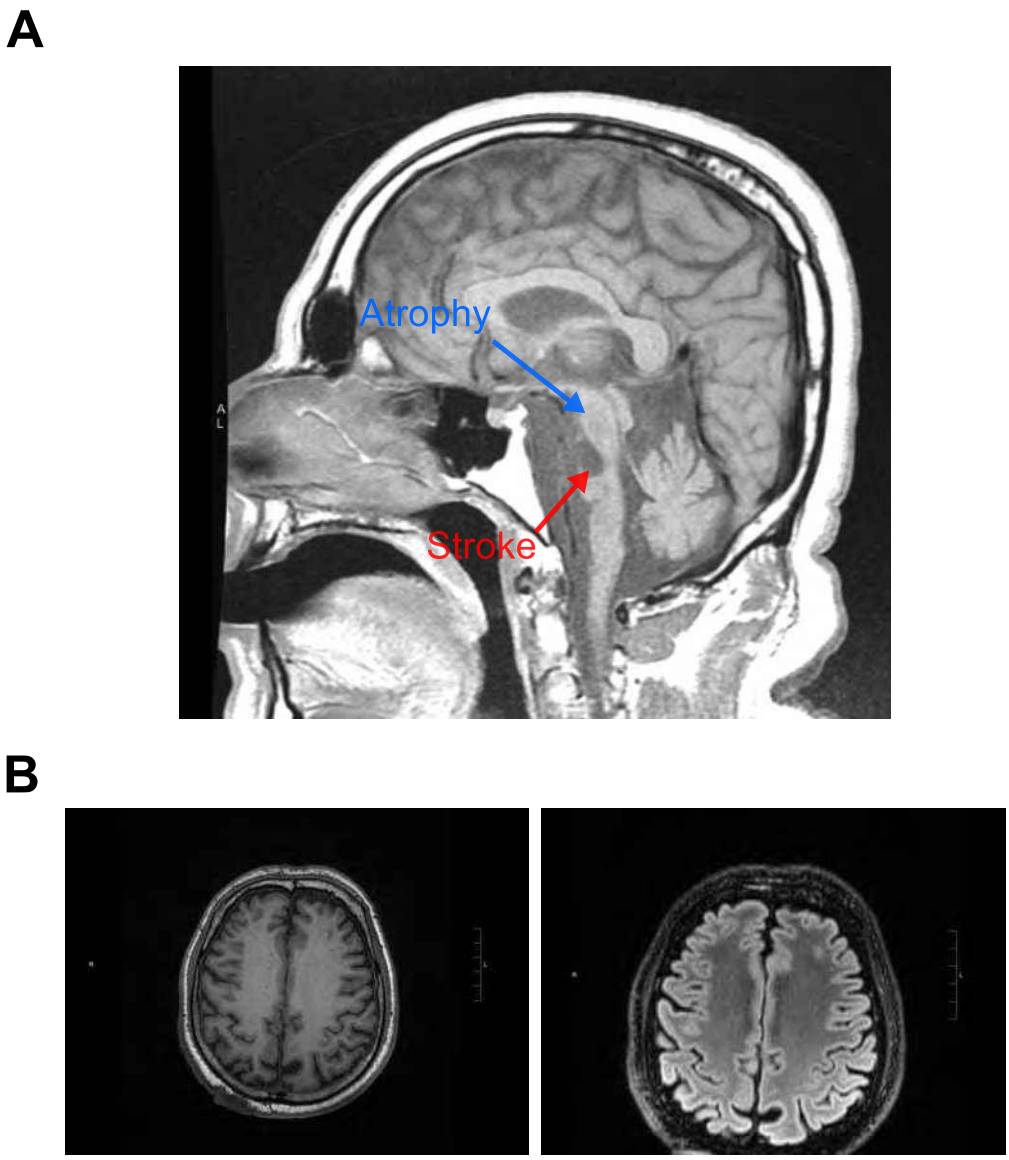

Figure S1. MRI results for the participant. Panel A shows a sagittal MRI for the participant, who has encephalomalacia and brain-stem atrophy (labeled in blue) caused by pontine stroke (labeled in red). Panel B shows two additional MRI scans that indicate the absence of cerebral atrophy, suggesting that cortical neuron populations (including those recorded from in this study) should be relatively unaffected by the participant's pathology.

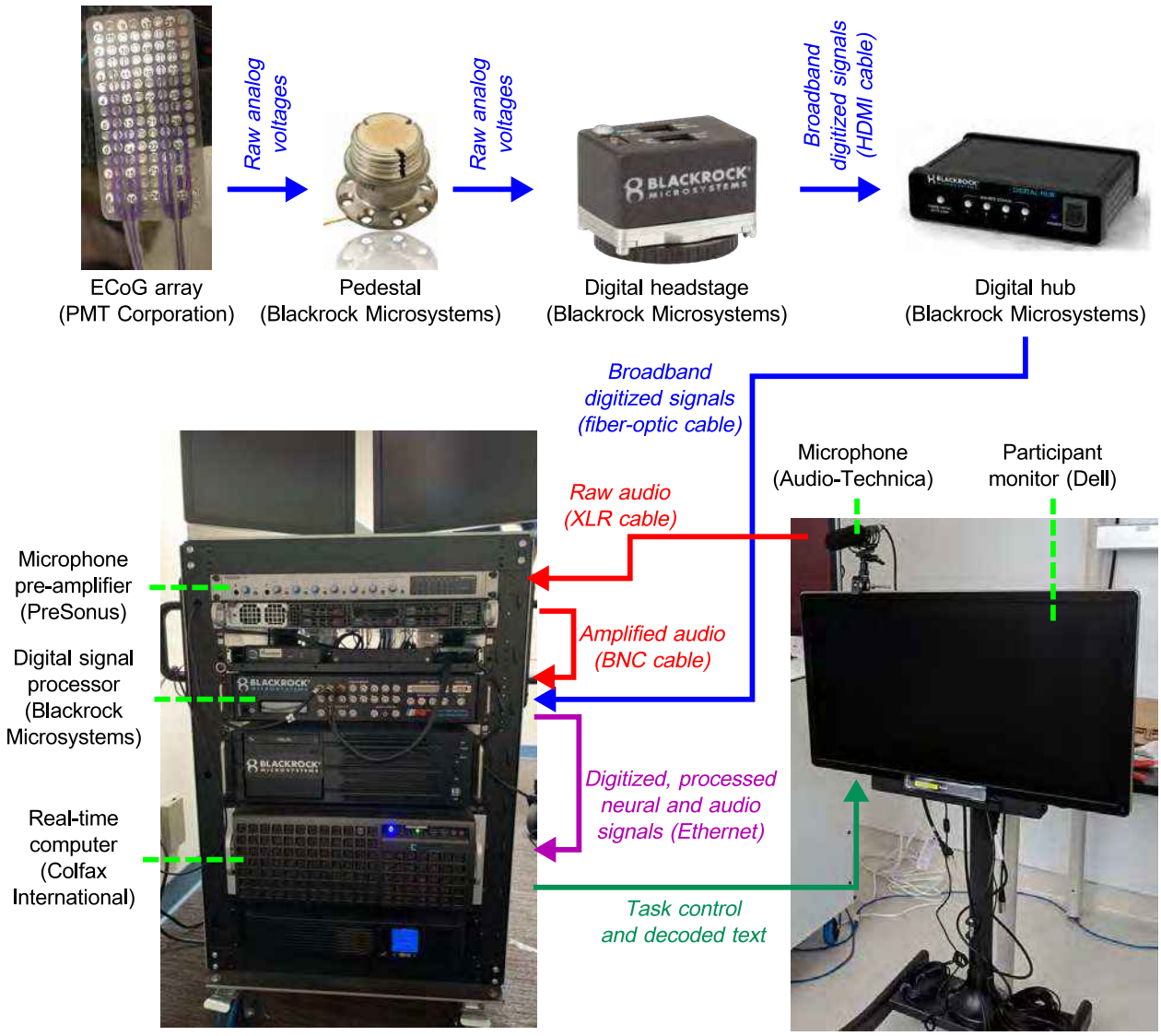

Figure S2. Real-time neural data acquisition hardware infrastructure. Electrocorticography (ECoG) data acquired from the implanted array and percutaneous pedestal connector are processed and transmitted to the Neuroport digital signal processor (DSP). Simultaneously, microphone data are acquired, amplified, and transmitted to the DSP. Signals from the DSP are transmitted to the real-time computer. The real-time computer controls the task displayed to the participant, including any decoded sentences that are provided in real time as feedback. Speaker data (output from the real-time computer) are also sent to the DSP and synchronized with the neural signals (not depicted). During earlier sessions, a human patient cable connected to the pedestal acquired the ECoG signals, which were then processed by a front-end amplifier before being transmitted to the DSP (the human patient cable and front-end amplifier, manufactured by Blackrock Microsystems, are not depicted here, but they replaced the digital headstage and digital hub in this pipeline when they were used).

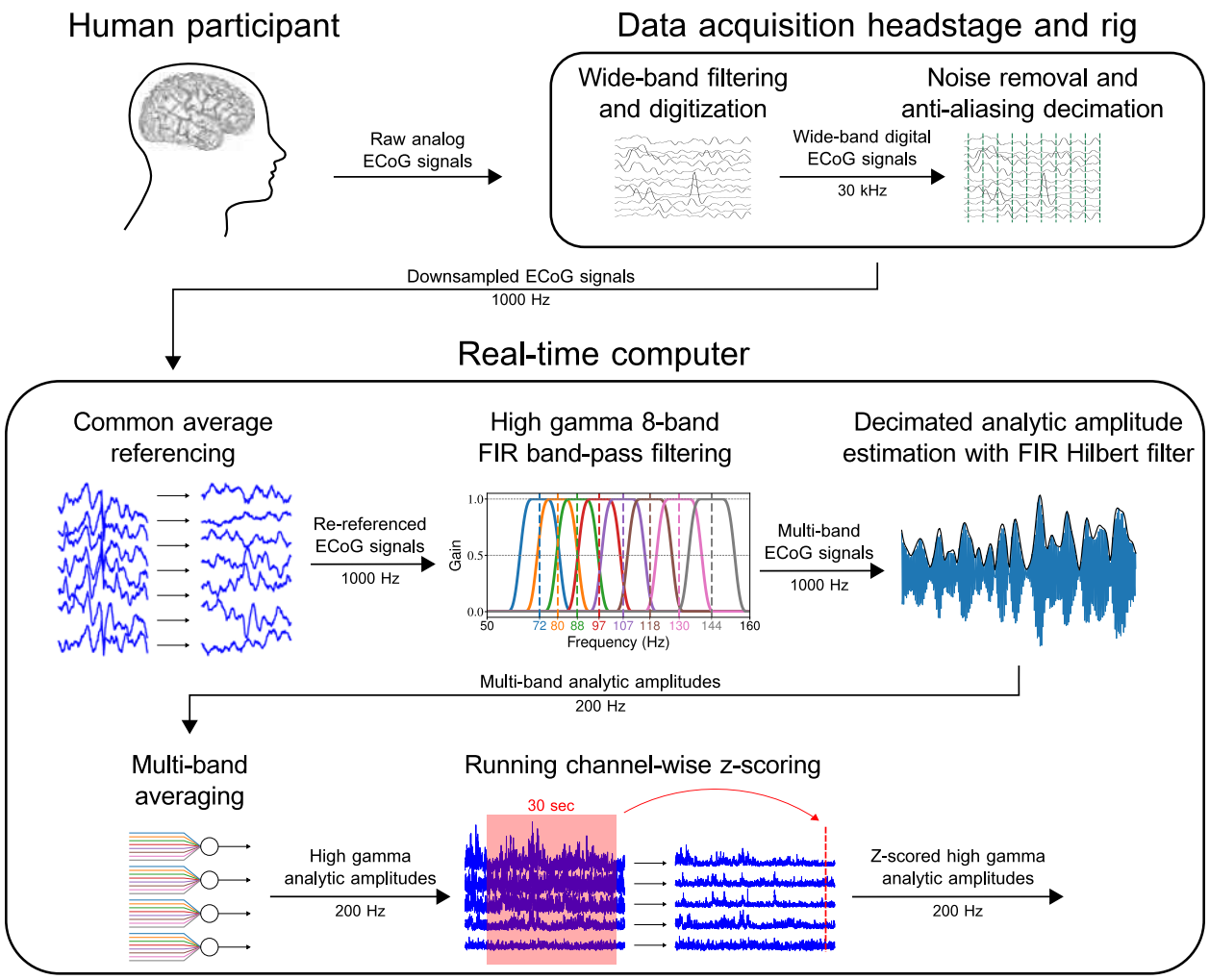

Figure S3. Real-time neural signal processing pipeline. Using the data acquisition headstage and rig, the participant's electrocorticography (ECoG) signals were acquired at 30 kHz, filtered with a wide-band filter, conditioned with a software-based line noise cancellation technique, low-pass filtered at 500 Hz, and streamed to the real-time computer at 1 kHz. On the real-time computer, custom software was used to perform common average referencing, multi-band high gamma band-pass filtering, analytic amplitude estimation, multi-band averaging, and running z-scoring on the ECoG signals. The resulting signals were then used as the measure of high gamma activity for the remaining analyses. This figure was adapted from our previous work [2], which implemented a similar neural signal preprocessing pipeline.

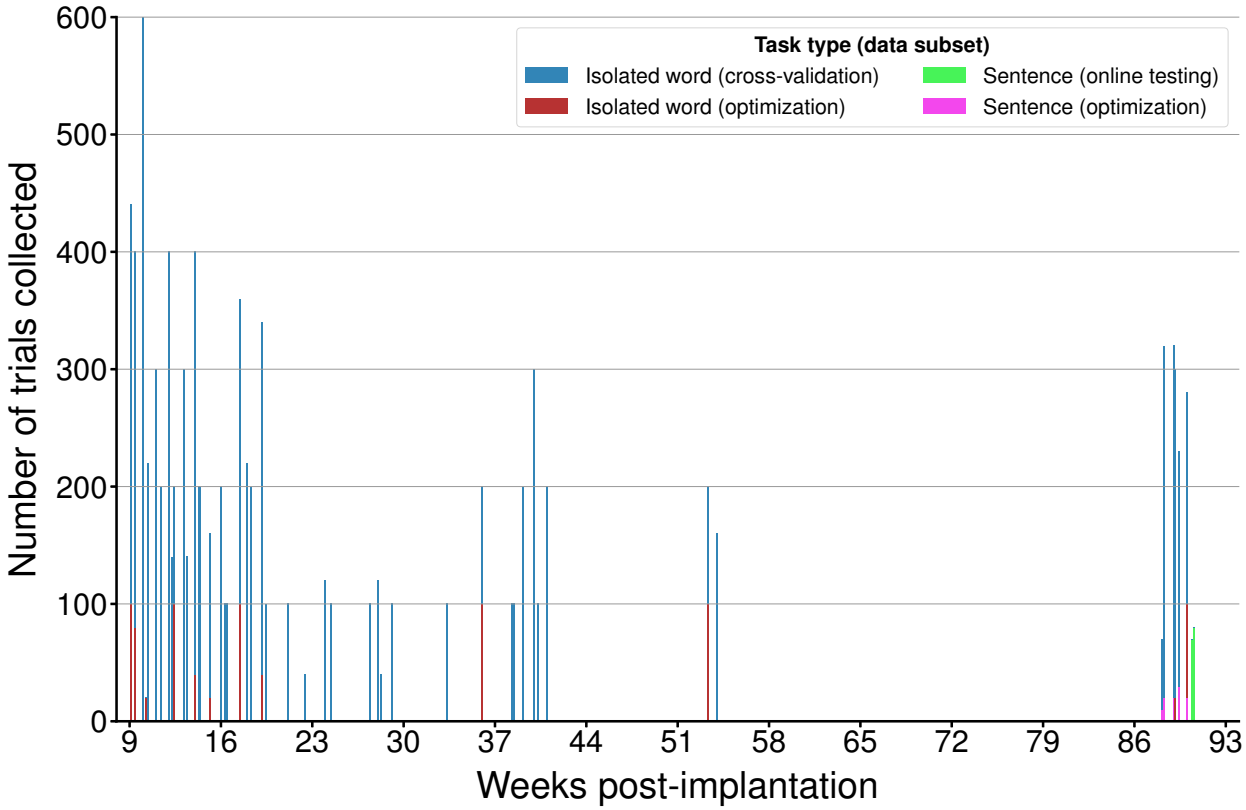

Figure S4. Data collection timeline. Bars are stacked vertically if more than one data type was collected in a day (the height of the stacked bars for any given day is equal to the total number of trials collected that day). The irregularity of the data collection schedule was influenced by external and clinical time constraints unrelated to the implanted device. The gap from 55–88 weeks was due to clinical guidelines concerning the COVID-19 pandemic.

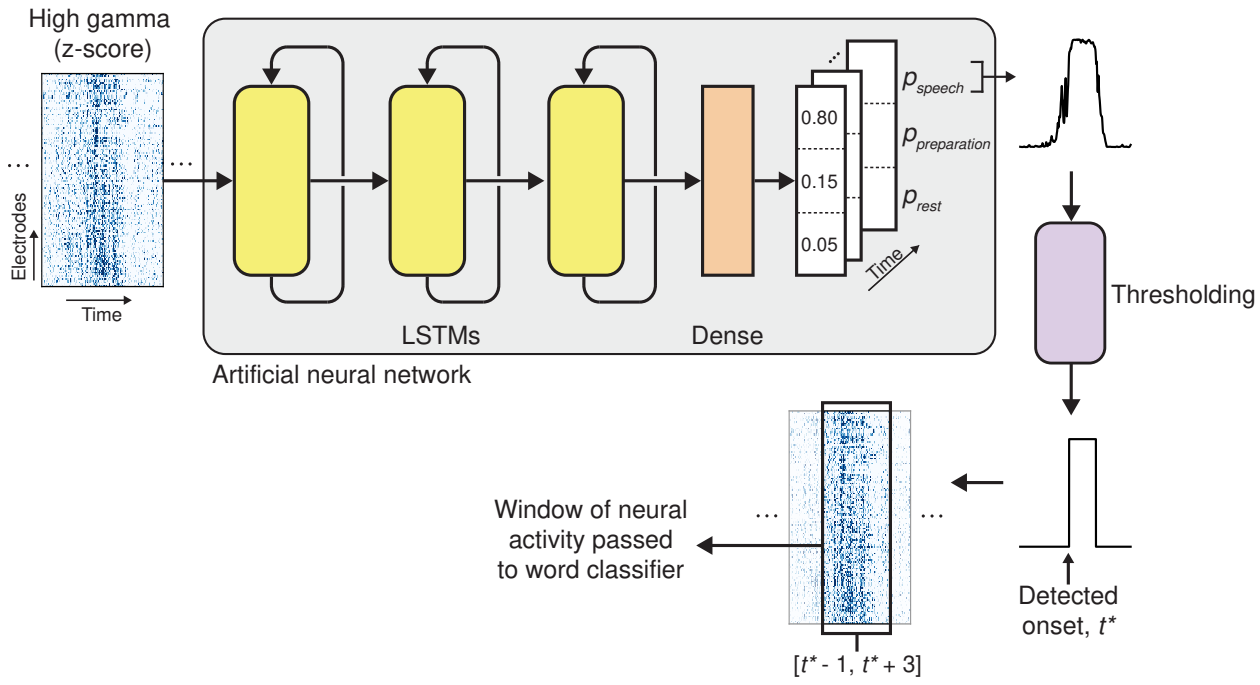

Figure S5. Speech detection model schematic. The z-scored high gamma activity across all electrodes is processed time point by time point by an artificial neural network consisting of a stack of three long short-term memory layers (LSTMs) and a single dense (fully connected) layer. The dense layer projects the latent dimensions of the last LSTM layer into probability space for three event classes: speech, preparation, and rest. The predicted speech event probability time series is smoothed and then thresholded with probability and time thresholds to yield onset (t^*) and offset times of detected speech events. During sentence decoding, each time a speech event was detected, the window of neural activity spanning from -1 to $+3$ seconds relative to the detected onset (t^*) was passed to the word classifier. The neural activity, predicted speech probability time series (upper right), and detected speech event (lower right) shown are the actual neural data and detection results across a 7-second time window for an isolated word trial in which the participant attempted to produce the word "family".

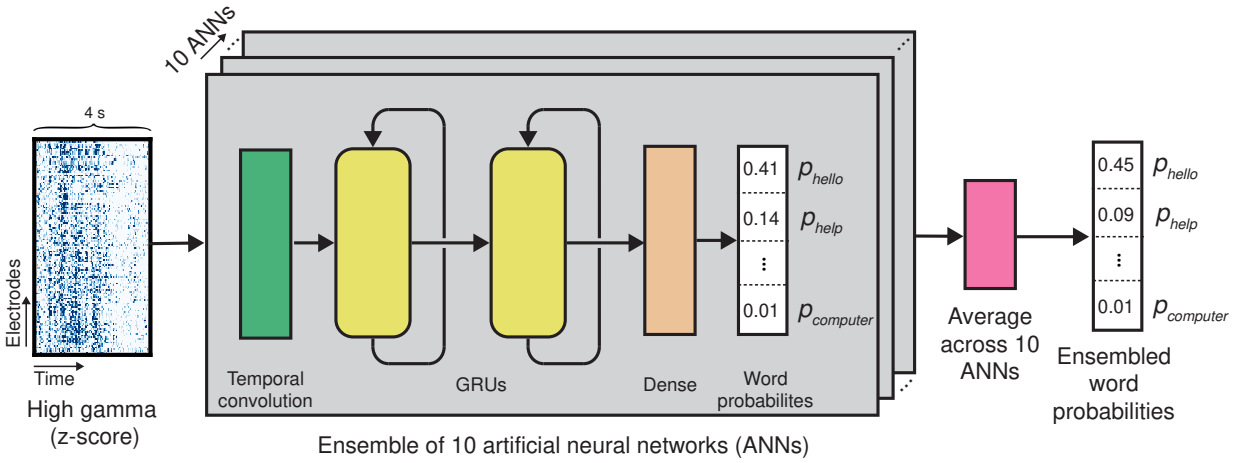

Figure S6. Word classification model schematic. For each classification, a 4-second time window of high gamma activity is processed by an ensemble of 10 artificial neural network (ANN) models. Within each ANN, the high gamma activity is processed by a temporal convolution followed by two bidirectional gated recurrent unit (GRU) layers. A dense layer projects the latent dimension from the final GRU layer into probability space, which contains the probability of each of the words from the 50-word set being the target word during the speech production attempt associated with the neural time window. The 10 probability distributions from the ensembled ANN models are averaged together to obtain the final vector of predicted word probabilities.

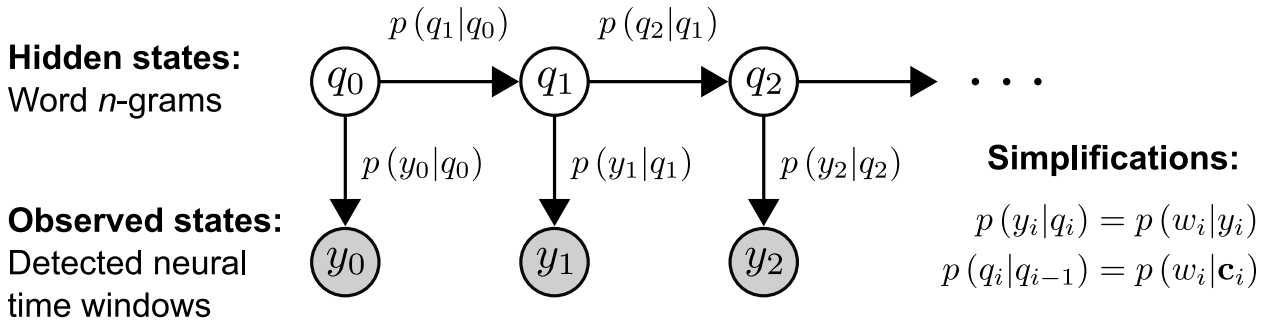

Figure S7. Sentence decoding hidden Markov model. This hidden Markov model (HMM) describes the relationship between the words that the participant attempts to produce (the hidden states q_i) and the associated detected time windows of neural activity (the observed states y_i). As described in Section S11, the HMM emission probabilities $p(y_0|q_0)$ can be simplified to $p(w_i|y_i)$ (the word likelihoods provided by the word classifier), and the HMM transition probabilities $p(q_i|q_{i-1})$ can be simplified to $p(w_i|\mathbf{c}_i)$ (the word-sequence prior probabilities provided by the language model).

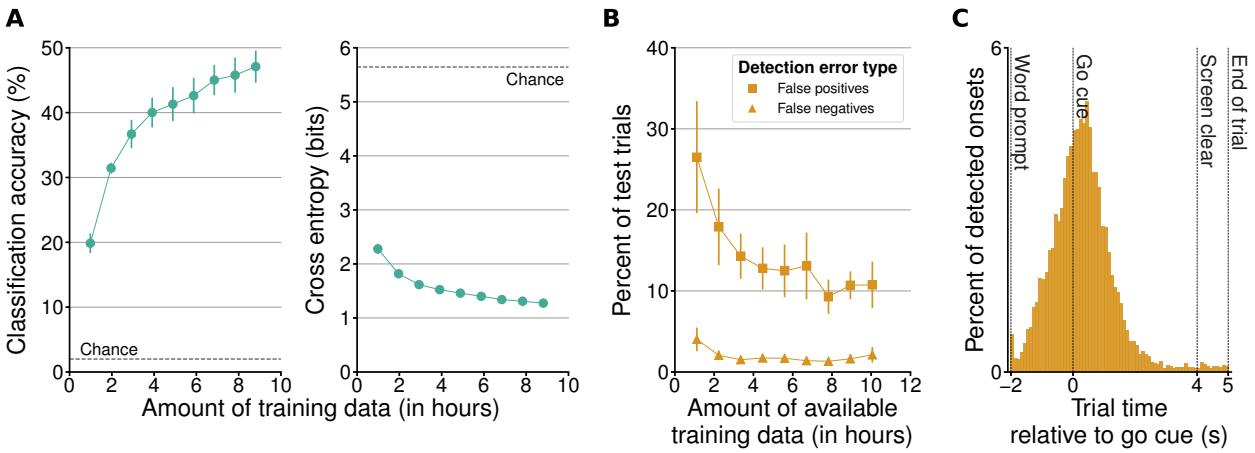

Figure S8. Auxiliary modeling results with isolated word data. Panel A shows the effect of the amount of training data on word classification accuracy (left) and cross-entropy loss (right) using cortical activity recorded during the participant's isolated word production attempts. Lower cross entropy indicates better performance. Each point depicts mean ± standard deviation across 10 cross-validation folds (the error bars in the cross-entropy plot were typically too small to be seen alongside the circular markers). Chance performance is depicted as a horizontal dashed line in each plot (chance cross-entropy loss is computed as the negative log (base 2) of the reciprocal of the number of word targets). Performance improved more rapidly for the first four hours of training data and then less rapidly for the next 5 hours, although it did not plateau. When using all available isolated word data, the information transfer rate was 25.1 bits per minute (not depicted), and the target word appeared in the top 5 predictions from the word classifier in 81.7% of trials (standard deviation was 2.1% across cross-validation folds; not depicted). Panel B shows the effect of the amount of training data on the frequency of detection errors during speech detection and detected event curation with the isolated word data. Lower error rates indicate better performance. False positives are detected events that were not associated with a word production attempt and false negatives are word production attempts that were not associated with a detected event. Each point depicts mean ± standard deviation across 10 cross-validation folds. As described in Section S13, not all of the available training data were used to fit each speech detection model, but each model always used between 47 and 83 minutes of data (not depicted). Panel C shows the distribution of onsets detected from neural activity across 9000 isolated word trials relative to the go cue (100 ms histogram bin size). This histogram was created using results from the final set of analyses in the learning curve scheme (in which all available trials were included in the cross-validated evaluation). The distribution of detected speech onsets had a mean of 308 ms after the associated go cues and a standard deviation of 1017 ms. This distribution was likely influenced to some degree by behavioral variability in the participant's response times. During detected event curation, 429 trials required curation to choose a detected event from multiple candidates (420 trials had 2 candidates and 9 trials had 3 candidates).

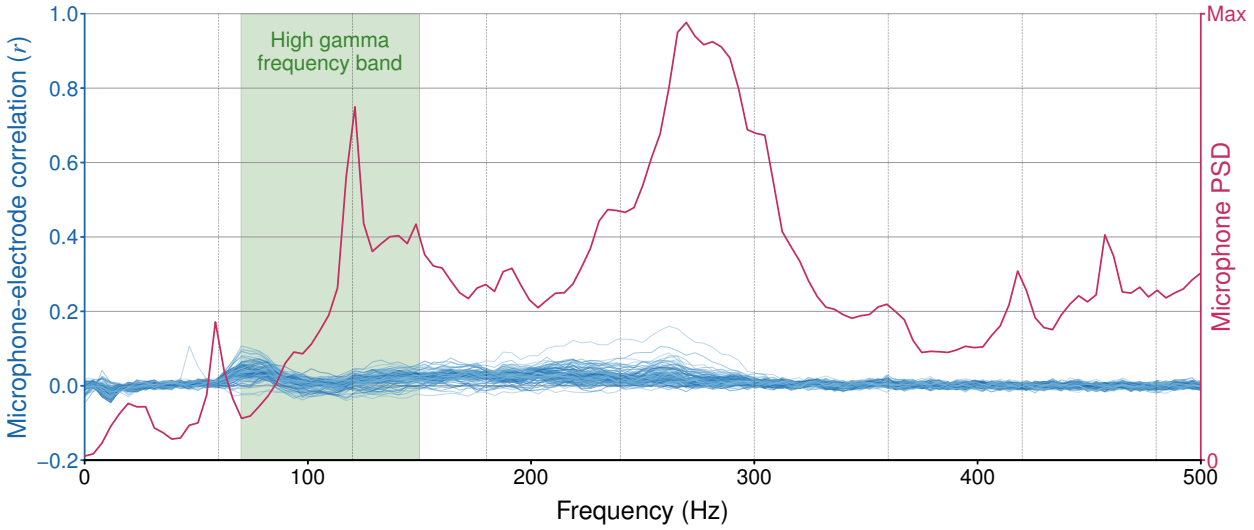

Figure S9. Acoustic contamination investigation. Each blue curve depicts the average correlations between the spectrograms from a single electrode and the corresponding spectrograms from the time-aligned microphone signal as a function of frequency. The red curve depicts the average power spectral density (PSD) of the microphone signal. Vertical dashed lines mark the 60 Hz line noise frequency and its harmonics. Highlighted in green is the high gamma frequency band (70–150 Hz), which was the frequency band from which we extracted the neural features used during decoding. Across all frequencies, correlations between the electrode and microphone signals are small. There is a slight increase in correlation in the lower end of the high gamma frequency range, but this increase in correlation occurs as the microphone PSD decreases. Because the correlations are low and do not increase or decrease with the microphone PSD, the observed correlations are likely due to factors other than acoustic contamination, such as shared electrical noise. After comparing these results to those observed in the study describing acoustic contamination (which informed the contamination analysis we used here) [39], we conclude that our decoding performance was not artificially improved by acoustic contamination of our electrophysiological recordings.

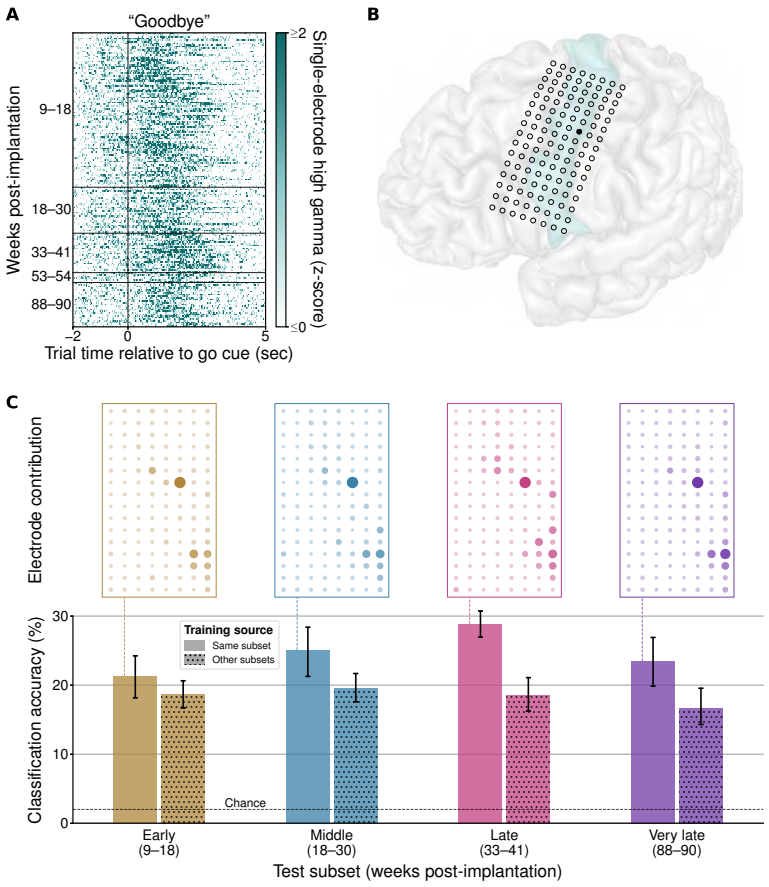

Figure S10. Long-term stability of speech-evoked signals. Panel A shows neural activity from a single electrode across all of the participant's attempts to say the word "Goodbye" during the isolated word task, spanning 81 weeks of recording. Panel B shows the participant's brain reconstruction overlaid with electrode locations. The electrode shown in Panel A is filled in with black. For anatomical reference, the precentral gyrus is highlighted in light green. Panel C shows word classification outcomes from training and testing the detector and classifier on subsets of isolated word data sampled from four non-overlapping date ranges. Each subset contains data from 20 attempted productions of each word. Each solid bar depicts results from cross-validated evaluation within a single subset, and each dotted bar depicts results from training on data from all of the subsets except for the one that is being evaluated. Each error bar shows the 95% confidence interval of the mean, computed across cross-validation folds. Chance accuracy is depicted as a horizontal dashed line. Electrode contributions computed during cross-validated evaluation within a single subset are shown on top (oriented with the most dorsal and posterior electrode in the upper-right corner). Plotted electrode size (area) and opacity are scaled by relative contribution. Each set of contributions is normalized to sum to 1. These results suggest that speech-evoked cortical responses remained relatively stable throughout the study period, although model recalibration every 2–3 months may still be beneficial for decoding performance.

Supplementary tables

Supplementary Table S1. Hyperparameter definitions and values.

Model	Hyperparameter description	Search space type	Value range	Optimal values[1]
Speech detector	Smoothing size	Uniform (integer)	$[1, 80]$	$(8, 5, 22)$
	Probability threshold	Uniform	$[0.1, 0.9]$	$(0.297, 0.319, 0.592)$
	Time threshold duration	Uniform (integer)	$[25, 150]$	$(79, 82, 93)$
Word classifier	Number of GRU layers	Uniform (integer)	$[1, 3]$	$(2, 2)$
	Nodes per GRU layer	Uniform (integer)	$[64, 512]$	$(434, 420)$
	Dropout fraction	Uniform	$[0.5, 0.95]$	$(0.704, 0.646)$
	Convolution kernel size and skip	Uniform (integer)	$[1, 2]$	$(2, 2)$
Language model	Initial word smoothing (ψ)	Logarithmically uniform	$[0.001, 1000]$	0.576
Viterbi decoder	Language model scaling factor (L)	Logarithmically uniform	$[0.1, 10]$	0.913

[1] For the speech detection hyperparameters, three values are listed: the first is the optimal value found when optimizing the detector on the isolated word optimization subset (used to detect word production attempts in the cross-validation subsets for evaluation by the word classifier), the second is the optimal value found when optimizing the detector on a subset of the pooled cross-validation subsets (used to detect word production attempts in the isolated word optimization subset for use during hyperparameter optimization of the word classifier), and the third is the optimal value found during hyperparameter optimization of the decoding pipeline with the sentence optimization subset (the value used during online sentence decoding). For the word classification hyperparameters, two values are listed: the first is the optimal value found when optimizing the classifier on the isolated word optimization subset (the value used for all isolated word evaluations) and the second is the optimal value found when optimizing the classifier on a small subset of isolated word trials near the end of the study period (the value used for offline sentence optimization and online sentence decoding). For the language modeling and Viterbi decoding hyperparameters, the optimal value listed was found when optimizing the decoding pipeline with the sentence optimization subset (the value used for online sentence decoding).

Supplementary references

1. Moses DA, Leonard MK, and Chang EF. Real-time classification of auditory sentences using evoked cortical activity in humans. Journal of Neural Engineering 2018;15:036005.
2. Moses DA, Leonard MK, Makin JG, and Chang EF. Real-time decoding of question-and-answer speech dialogue using human cortical activity. Nature Communications 2019;10.
3. Ludwig KA, Miriani RM, Langhals NB, Joseph MD, Anderson DJ, and Kipke DR. Using a common average reference to improve cortical neuron recordings from microelectrode arrays. Journal of neurophysiology 2009;101:1679–89.
4. Williams AJ, Trumpis M, Bent B, Chiang CH, and Viventi J. A Novel μECoG Electrode Interface for Comparison of Local and Common Averaged Referenced Signals. In: 2018 40th Annual International Conference of the IEEE Engineering in Medicine and Biology Society (EMBC). Honolulu, HI: IEEE, 2018:5057–60.
5. Parks TW and McClellan JH. Chebyshev Approximation for Nonrecursive Digital Filters with Linear Phase. IEEE Transactions on Circuit Theory 1972;19:189–94.
6. Romero DET and Jovanovic G. Digital FIR Hilbert Transformers: Fundamentals and Efficient Design Methods. In: MATLAB - A Fundamental Tool for Scientific Computing and Engineering Applications - Volume 1. 2012:445–82.
7. Welford BP. Note on a Method for Calculating Corrected Sums of Squares and Products. Technometrics 1962;4:419–9.
8. Weiss JM, Gaunt RA, Franklin R, Boninger ML, and Collinger JL. Demonstration of a portable intracortical brain-computer interface. Brain-Computer Interfaces 2019;6:106–17.
9. Bergstra J, Yamins DLK, and Cox DD. Making a Science of Model Search: Hyperparameter Optimization in Hundreds of Dimensions for Vision Architectures. Icml 2013:115–23.
10. Liaw R, Liang E, Nishihara R, Moritz P, Gonzalez JE, and Stoica I. Tune: A Research Platform for Distributed Model Selection and Training. arXiv:1807.05118 2018.
11. Li L, Jamieson K, Rostamizadeh A, et al. A System for Massively Parallel Hyperparameter Tuning. arXiv:1810.05934 2020.
12. Paszke A, Gross S, Massa F, et al. PyTorch: An Imperative Style, High-Performance Deep Learning Library. In: Advances in Neural Information Processing Systems 32. Ed. by Wallach H, Larochelle H, Beygelzimer A, d'Alché-Buc F, Fox E, and Garnett R. Curran Associates, Inc., 2019:8024–35.
13. Hochreiter S and Schmidhuber J. Long Short-Term Memory. Neural Computation 1997;9:1735–80.
14. Dash D, Ferrari P, Dutta S, and Wang J. NeuroVAD: Real-Time Voice Activity Detection from Non-Invasive Neuromagnetic Signals. Sensors 2020;20:2248.
15. Werbos P. Backpropagation through time: what it does and how to do it. Proceedings of the IEEE 1990;78:1550–60.

16. Elman JL. Finding Structure in Time. Cognitive Science 1990;14:179–211.

17. Williams RJ and Peng J. An Efficient Gradient-Based Algorithm for On-Line Training of Recurrent Network Trajectories. Neural Computation 1990;2:490–501.

18. Kingma DP and Ba J. Adam: A Method for Stochastic Optimization. arXiv:1412.6980 2017.

19. Krizhevsky A, Sutskever I, and Hinton GE. ImageNet Classification with Deep Convolutional Neural Networks. In: Advances in Neural Information Processing Systems 25. Ed. by Pereira F, Burges CJC, Bottou L, and Weinberger KQ. Curran Associates, Inc., 2012:1097–105.

20. Makin JG, Moses DA, and Chang EF. Machine translation of cortical activity to text with an encoder–decoder framework. Nature Neuroscience 2020;23:575–82.

21. Virtanen P, Gommers R, Oliphant TE, et al. SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python. Nature Methods 2020;17:261–72.

22. Martín Abadi, Ashish Agarwal, Paul Barham, et al. TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems. Software available from tensorflow.org. 2015.

23. Zhang Y, Chan W, and Jaitly N. Very Deep Convolutional Networks for End-to-End Speech Recognition. arXiv:1610.03022 2016.

24. Cho K, Merrienboer B van, Gulcehre C, et al. Learning Phrase Representations using RNN Encoder-Decoder for Statistical Machine Translation. arXiv:1406.1078 2014.

25. Pascanu R, Mikolov T, and Bengio Y. On the difficulty of training recurrent neural networks. In: Proceedings of the 30th International Conference on Machine Learning. Ed. by Dasgupta S and McAllester D. Vol. 28. Proceedings of Machine Learning Research. Atlanta, Georgia, USA: PMLR, 2013:1310–8.

26. Sollich P and Krogh A. Learning with ensembles: How overfitting can be useful. In: Advances in Neural Information Processing Systems 8. Ed. by Touretzky DS, Mozer MC, and Hasselmo ME. MIT Press, 1996:190–6.

27. Chen SF and Goodman J. An empirical study of smoothing techniques for language modeling. Computer Speech & Language 1999;13:359–93.

28. Kneser R and Ney H. Improved backing-off for M-gram language modeling. In: 1995 International Conference on Acoustics, Speech, and Signal Processing. Vol. 1. Detroit, MI, USA: IEEE, 1995:181–4.

29. Bird S, Klein E, and Loper E. Natural language processing with Python: analyzing text with the natural language toolkit. O'Reilly Media, Inc., 2009.

30. Group TH. Hierarchical Data Format. 1997.

31. Collette A. Python and HDF5: unlocking scientific data. "O'Reilly Media, Inc.", 2013.

32. Heafield K. KenLM: Faster and Smaller Language Model Queries. In: Proceedings of the Sixth Workshop on Statistical Machine Translation. WMT '11. Association for Computational Linguistics, 2011:187–97.

33. Viterbi AJ. Error Bounds for Convolutional Codes and an Asymptotically Optimum Decoding Algorithm. IEEE Transactions on Information Theory 1967;13:260–9.

34. Jurafsky D and Martin JH. Speech and language processing: an introduction to natural language processing, computational linguistics, and speech recognition. 2nd. Upper Saddle River, New Jersey: Pearson Education, Inc., 2009.

35. Simonyan K, Vedaldi A, and Zisserman A. Deep Inside Convolutional Networks: Visualising Image Classification Models and Saliency Maps. In: Workshop at the International Conference on Learning Representations. Ed. by Bengio Y and LeCun Y. Banff, Canada, 2014.

36. Wolpaw JR, Birbaumer N, McFarland DJ, Pfurtscheller G, and Vaughan TM. Brain-computer interfaces for communication and control. Clinical neurophysiology : official journal of the International Federation of Clinical Neurophysiology 2002;113:767–91.

37. Mugler EM, Patton JL, Flint RD, et al. Direct classification of all American English phonemes using signals from functional speech motor cortex. Journal of neural engineering 2014;11:35015–15.

38. Speier W, Arnold C, and Pouratian N. Evaluating True BCI Communication Rate through Mutual Information and Language Models. PLoS ONE 2013;8. Ed. by Wennekers T:e78432.

39. Roussel P, Godais GL, Bocquelet F, et al. Observation and assessment of acoustic contamination of electrophysiological brain signals during speech production and sound perception. Journal of Neural Engineering 2020;17:056028.