EAI 820 · SPEECH AND LANGUAGE TECHNOLOGIES

1 Introduction

An overview of the course

MSc Embedded Artificial Intelligence · elective · 3 units

This is an introduction to EAI 820 — Speech and Language Technologies, an elective in the MSc programme in Embedded Artificial Intelligence. Its purpose is to set out what the subject is, why it is difficult, what vocabulary you will need, and how the topics named in the course outline fit together into a coherent whole.

The course outline lists its contents as a sequence of technologies: speech synthesis and the machinery behind it; speech recognition; language recognition; automatic translation, summarizers, spell checkers and grammar checkers; and the development of mobile speech and language applications. Read as a list, these look like eight or nine separate subjects. **They are not.** They rest on a small set of shared ideas, and a good deal of this chapter is spent identifying those ideas, so that the rest of the course can proceed by variation rather than by starting over each week.

What you should be able to do after this chapter

1. Describe the speech and language loop, and place any named technology within it.
2. State the three properties of human language that defeat naive approaches, with an example of each.
3. Name the levels of linguistic description and give an example of an ambiguity at each.
4. Explain the six ideas shared by every system in this course, and recognise them inside an unfamiliar system.
5. Distinguish a phoneme from an allophone, and lexical stress from pitch accent.
6. Match an evaluation metric to the task it belongs to.
7. Explain why an always-on speech interface on a device is built as a cascade.

1 What This Course Is About

Language is the interface humans were built for. We acquire it without instruction, use it without effort, and route almost all of our knowledge through it. It is therefore unsurprising that getting machines to handle language has been a goal of computing since before there were computers to speak of — and that it has turned out to be one of the hardest things we have ever asked machines to do.

The field has two historically distinct halves that have now largely merged. **Natural language processing** deals with language as text: segmenting it, analysing it, translating it, generating it. **Speech processing** deals with language as sound: recognising it, synthesising it, identifying who is

speaking and in what language. For fifty years these were separate communities with separate conferences and separate mathematics. Today they share an architecture, a training strategy, and increasingly a single model.

For a student of embedded artificial intelligence, the convergence matters for a specific and practical reason. A single architecture that handles audio, text and their combination is an architecture you can compress, quantise and deploy once, rather than a collection of task-specific components each needing its own optimisation. The engineering question throughout this course is therefore not only "how does this work?" but *"what does it cost, and can it run on the device in front of me?"* That second question is what distinguishes EAI 820 from a general course in language technology, and it is the question the final weeks are built around.

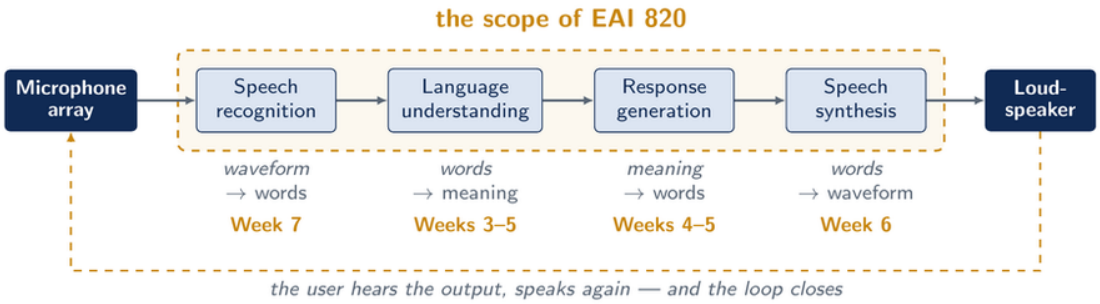


Figure 1.1. *The human–machine language loop, with the week of EAI 820 in which each stage is treated. This course covers the four inner stages; the outer two are hardware.*

Figure 1.1 shows the loop that organises the whole subject. Sound enters, becomes words, becomes meaning, becomes new words, becomes sound again. Almost every technology named in the course outline is a component of that loop, a shortcut across it, or a variation on it. A dictation application runs only the left half. A screen reader runs only the right half. A speech-to-speech translator runs the whole loop with the language changing in the middle. A wake-word detector is a tiny, cheap, always-listening fragment of the leftmost stage whose only job is to decide when to wake the rest of the system up.

Keep the loop in mind as a filing system. When a new technique appears in a later week, the first question worth asking is which stage it belongs to and what it consumes and produces.

2 Why Language Is Hard

Before looking at solutions it is worth being precise about the problem. Three properties of human language conspire to defeat approaches that would work for almost any other kind of data.

Ambiguity

Almost every level of linguistic structure is ambiguous, and the ambiguities multiply. Consider the spoken sentence *I saw her duck*. As sound, it is ambiguous even in its word boundaries — speech has no spaces, so deciding where one word ends and the next begins is itself a decision the system must make. As a string of words it is ambiguous between at least four readings, depending on whether *duck* is a noun or a verb, whether *her* is a possessive or an object pronoun, and whether *saw* means perceived or cut with a saw.

Worked example: ambiguity at every level

Input: *I saw her duck.*

Acoustic — Where does one word end and the next begin? There is no silence between them.

Lexical — duck is the bird (noun) or the action of lowering the head (verb)?

Syntactic — Is her duck a noun phrase, or is her the subject of a reduced clause?

Semantic — saw is the past tense of see, or the present tense of to saw?

Pragmatic — Who is her? Resolving this requires the preceding few sentences.

Four binary decisions give up to sixteen combinations; a listener picks one in a few hundred milliseconds and is almost never wrong. That effortlessness is what we are trying to reproduce.

Two consequences follow, and both shape the design of every system in this course. The first is that a system handling only one level of structure will fail on the others. The second is that the levels interact: resolving the syntax often depends on having resolved the semantics, and the other way round. A pipeline that commits early to one level's answer propagates that answer's errors into every later stage.

Variation

The same word is never produced twice in exactly the same way. Acoustically, the vowel in *cat* differs between speakers, between a whisper and a shout, between a quiet room and a noisy street, and even between two utterances by the same speaker one second apart. Orthographically, users write *tomorrow, tomorow, 2moro* and *tmrw*. Dialectally, a system trained on one variety of a language degrades sharply on another — a fairness problem as much as an engineering one, and one that matters acutely in a multilingual country where most users are speaking a second or third language.

Productivity and the long tail

Language is generative: speakers routinely produce sentences that have never been uttered before, and coin words that have never been written before. Any system built on a fixed list of words will meet a word that is not on the list. The distribution of words is steeply skewed: a handful of words are extremely frequent, and an unbounded tail of words occurs once or never. Two of the techniques introduced in the first weeks of this course — **subword tokenization** and **smoothing** — exist precisely to handle events that were never seen in training. The first ensures that any string whatsoever can be represented, by falling back to smaller units. The second ensures that nothing is assigned probability zero merely because it did not occur in the data we happened to collect.

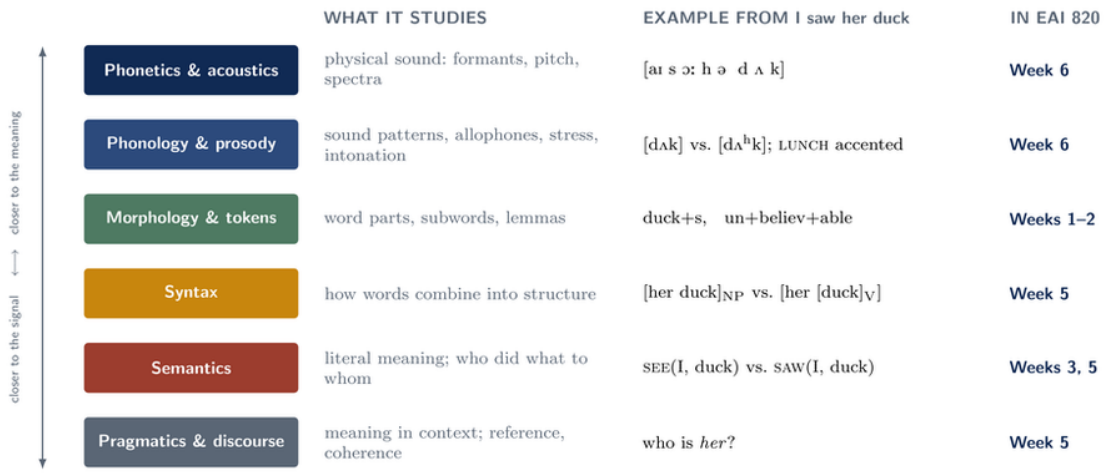


Figure 1.2. *The levels of linguistic description, with the week of EAI 820 in which each is treated. Classical systems built one module per level and chained them together; modern systems learn a single representation that implicitly encodes several levels at once — but this vocabulary remains indispensable for describing what a system does and where it fails.*

Figure 1.2 lays out the levels of linguistic description that give the subject its vocabulary. Two remarks are worth making. First, the levels are *descriptive*, not architectural: a modern speech recogniser contains no module labelled "phonology", yet phonological vocabulary is still how we describe the errors it makes. Second, the course outline is written largely in the language of levels — "allophones", "morphology", "stress and intonation" — and you will need these terms to read the literature even where the systems you build do not contain matching components.

3 The Shared Toolkit

The technologies named in the course outline look, at first sight, like separate subjects. Underneath, they are six ideas in different arrangements. This section states each in a paragraph or two. The mathematics comes in the weeks that follow; what matters here is that you should be able to see these six ideas inside any system you meet later, including systems that had not been invented when this chapter was written.

Idea 1 — Turn the input into a sequence of tokens

Neither text nor audio arrives in a form a computer model can consume. Text arrives as characters; audio arrives as a sequence of amplitude measurements, typically sixteen thousand of them per second. Both must be converted into a sequence drawn from a finite vocabulary of **tokens**.

For text, the standard method is **byte-pair encoding**, which learns a vocabulary of subwords by starting from individual characters and repeatedly merging the most frequent adjacent pair. Common words end up as single tokens; rare words are split into pieces. This solves the unknown-word problem: any string whatsoever can be represented, because the character level is always available as a fallback.

For audio, the parallel idea is the **neural audio codec**, which learns a vocabulary of discrete audio tokens by training an encoder to compress a waveform into a short sequence of symbols and a decoder to reconstruct it. The intellectual move is identical — turn a continuous or open-ended signal into a finite alphabet — and once it has been made, the same modelling machinery applies to sound as to text. This single observation is what merged two fields that had kept separate conferences for

half a century, and it is why a course can treat speech synthesis and machine translation as relatives rather than strangers.

Idea 2 — Represent tokens as vectors

A token identifier is an arbitrary integer; it carries no information about meaning. **Embeddings** are dense vectors, typically a few hundred to a few thousand numbers each, learned so that tokens appearing in similar contexts receive similar vectors. The governing principle is the **distributional hypothesis** — words that occur in the same environments tend to mean similar things — and the standard measure of similarity between two vectors is the **cosine** of the angle between them.

> **Cosine similarity**
>
> For two vectors u and v, the cosine similarity is their dot product divided by the product of their lengths. Values run from −1 to 1, or from 0 to 1 for non-negative count vectors. Cosine ignores vector length, which matters because a word's raw count vector is longer simply if the word is frequent. You will use this in Week 3 to find a word's nearest neighbours, in Week 5 to rank sentences for a summarizer, and again in Week 6 to score how closely a synthesised voice matches a target speaker.

Idea 3 — Predict the next token

A **language model** assigns a probability to the next token given the tokens so far. The simplest version approximates the history by the previous few tokens and estimates the probabilities by counting occurrences in a large body of text. It is obsolete as a production technology but indispensable as a teaching device, because it makes the objective completely transparent before any neural machinery is introduced. Week 2 builds one from scratch.

Language models are evaluated by **perplexity**, the inverse geometric mean of the probability the model assigned to a held-out test set. Lower is better; a perplexity of one hundred means the model is on average as uncertain as if it were choosing uniformly among a hundred words. Perplexity is only comparable across models that share a vocabulary and a tokenization — a trap worth remembering when reading published results.

The move from counting to neural estimation changes the estimator, not the objective. This is why **conditional generation** — producing a sequence one token at a time, each conditioned on everything before it — is the single most reusable technique in this course. Four of the technologies in the course outline are conditional generation with different material on the left of the conditioning bar:

Technology	Conditioned on
Automatic translation	the source sentence
Summarization	the source document
Speech recognition	a sequence of audio feature vectors
Speech synthesis	the text, plus a sample of the desired voice

Idea 4 — Learn by gradient descent on a loss

Supervised learning supplies the fourth idea and the whole training apparatus that goes with it: a function with adjustable parameters, a **loss function** measuring disagreement between the function's output and the training labels, and **gradient descent** to adjust the parameters so as to

reduce that disagreement. Replacing the function with a stack of nonlinear layers gives a neural network, and **backpropagation** computes the gradients efficiently through the stack. Nothing about this training loop changes when the model grows to a hundred billion parameters; only the scale of the engineering does.

Idea 5 — Attention

The fifth idea is the one that makes the others practical at scale. In a sequence, the information needed to interpret one position may sit anywhere — one position back, or two thousand. **Attention** lets every position build its representation as a weighted average of the representations of other positions, where the weights are computed from the content itself. Each position emits a query; every position offers a key and a value; the query is compared against all the keys, the comparisons are normalised into weights, and the values are averaged accordingly.

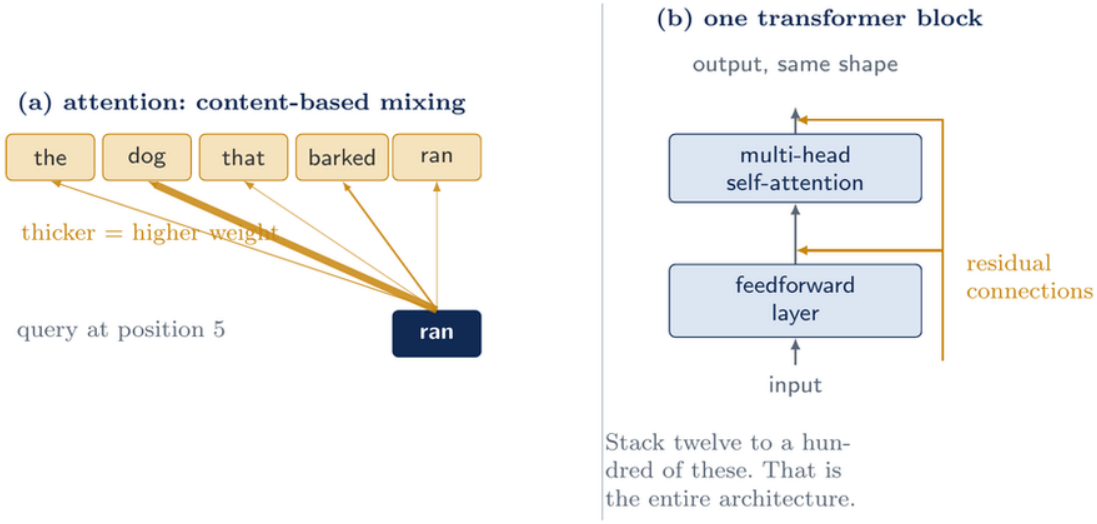


Figure 1.3. *Attention and the transformer block that packages it. Attention lets a word draw on another several positions away with no intervening machinery; the block that stacks it — attention plus a feedforward layer with residual connections — is the repeating unit of every model in this course.*

Figure 1.3 shows both the mechanism and the block — the **transformer** block — that packages it. Attention is what allows a speech encoder to relate a frame of audio to a frame half a second away, and what allows a translation system to look back at the relevant part of the source sentence when producing each target word. It is the same operation in both cases. Week 4 is devoted to it, and you will implement it directly.

Idea 6 — Pretrain, then adapt

The last structural idea is that the expensive general learning and the cheap task-specific learning should be separated. A model is **pretrained** on an enormous quantity of unlabelled data, using an objective that requires no human annotation — predict the next token, predict a hidden token, predict a masked span of audio. It is then **adapted** to a particular task with a comparatively small quantity of labelled data or instructions, by further training, by instruction tuning, or simply by being prompted.

This is the economic structure of the field as much as its technical structure. The pretraining is done once, at great cost, by whoever can afford it; the adaptation is done many times, by everyone else.

Most practitioners never pretrain anything. For an embedded systems engineer a third stage matters just as much: taking an adapted model and compressing it until it fits the target device, which is the subject of Section 7 and of Week 8.

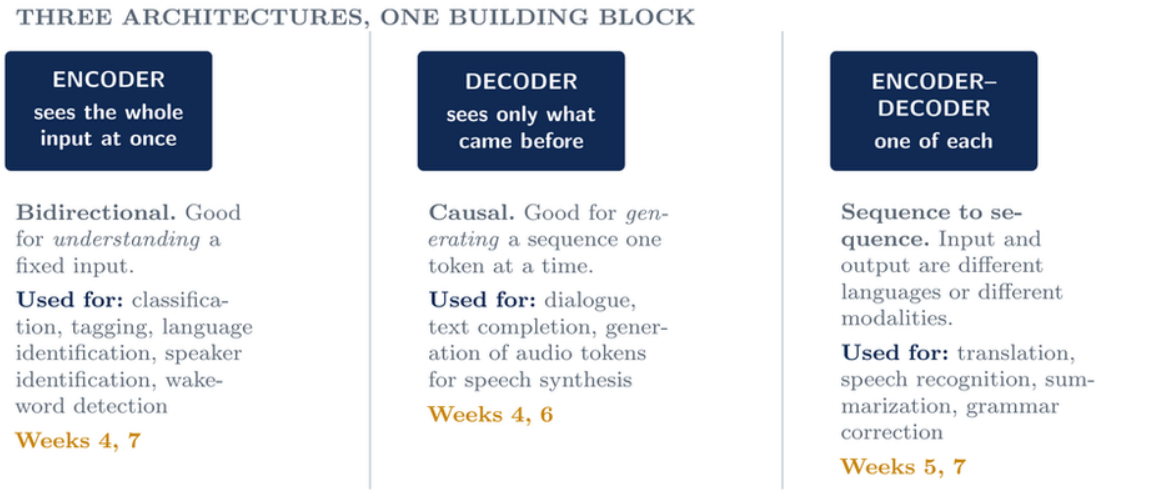


Figure 1.4. *The three ways the transformer block gets wired up. Choosing among them is the first design decision in any project, and it follows directly from whether the input is fixed, the output is generated, or both.*

Figure 1.4 summarises the three configurations you will meet. Almost every named system in this field is one of these three with a particular pretraining objective attached. Being able to place an unfamiliar system into this taxonomy on first reading is a genuinely useful professional skill, and a reasonable thing to expect of yourself by the end of the course.

> **Laboratory work, Weeks 1–4**
>
> The first four laboratory sessions instantiate Ideas 1 to 5 in plain Python on a CPU, with no model downloads and no network access required: implementing byte-pair merges by hand; building a counting language model with smoothing and measuring its perplexity; computing cosine similarity over a small term–document matrix; and implementing scaled dot-product attention in NumPy. These run offline on any laptop. Work that needs a GPU begins in Week 6.

4 Speech Synthesis

Speech synthesis, or text-to-speech, is the task of mapping a string of text to an acoustic waveform in a specified voice. It heads the course outline, and it carries with it four of the outline's other entries — allophones, pronunciation dictionaries, morphological and rule-based letter-to-sound conversion, and stress and intonation — all of which are components of the synthesis problem. This section introduces them together, since they only make sense in relation to one another. Week 6 treats them in full.

Synthesis has a longer history than any other topic in this course. In Vienna in the 1770s Wolfgang von Kempelen built a mechanical speaking machine with a bellows for lungs, a reed for vocal folds, and a flexible leather vocal tract that an operator manipulated by hand to produce consonants and vowels. We no longer build synthesisers out of wood and leather, but the problem he was solving is recognisably the one we still solve, and his machine's division into a source of sound and a filter that shapes it survives in the vocabulary of the field.

Two architectures, decades apart

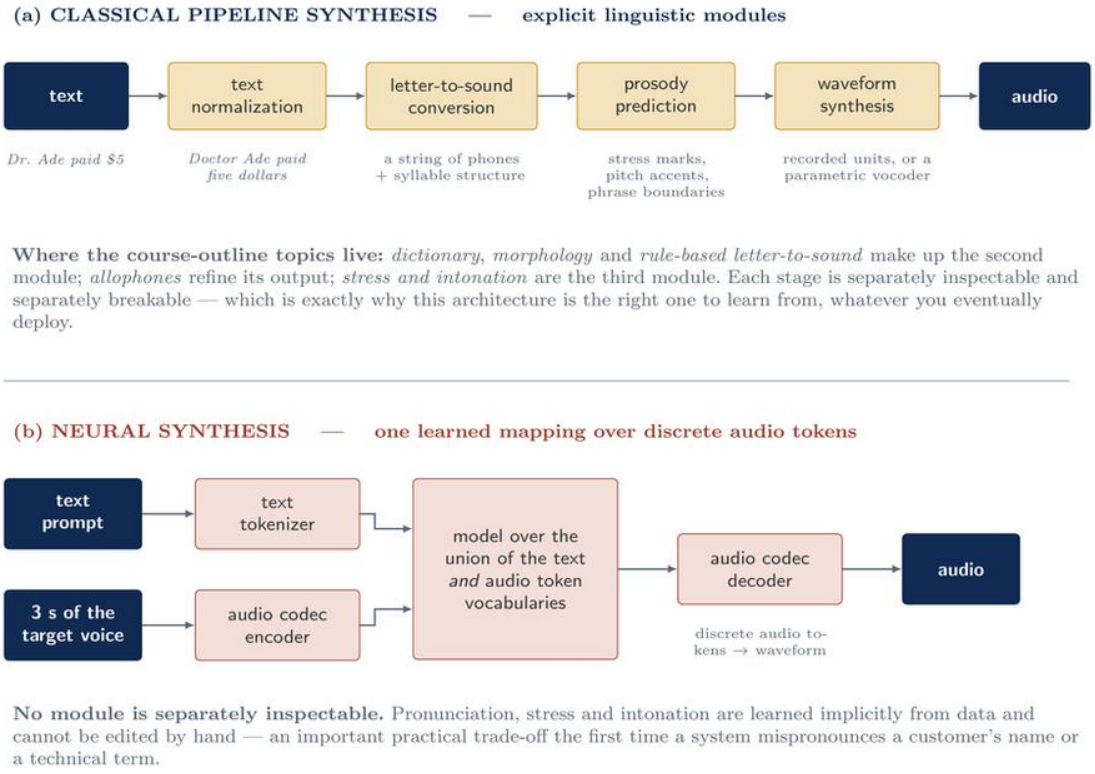


Figure 1.5. *Two generations of speech synthesis. The classical pipeline (a) makes every linguistic decision explicit; the neural approach (b) collapses the pipeline into conditional generation over discrete audio tokens. Production systems are of type (b), but the vocabulary of type (a) remains how the field describes what goes wrong.*

Figure 1.5 contrasts the two designs you will work with. The classical pipeline decomposes synthesis into four modules chained together. The neural approach treats it as conditional generation: tokenize the text, tokenize a short sample of the desired voice with an audio codec, and generate the audio tokens for that text in that voice. Because the voice is supplied at inference time rather than fixed during training, this is called **zero-shot synthesis** — the target speaker may never have appeared in the training data at all.

Both belong in this course, for different reasons. The neural approach is what you will deploy. The classical pipeline is what you must understand in order to diagnose failures, to build for a language with no large recorded corpus, and to make sense of the terminology in the course outline.

Text normalization

The first module handles everything that is written but not said as written: numbers, dates, currency, abbreviations, addresses, web addresses. *$5* is *five dollars*; *Dr.* is *Doctor* before a name but *Drive* after one; *1985* is *nineteen eighty-five* as a year but a quantity otherwise. Deciding which reading applies is a classification problem over non-standard words. It is unglamorous, and it is where a disproportionate share of user-visible synthesis errors originate.

From letters to sounds

The second module converts spelling into a sequence of **phones** — speech sounds. Two notations are standard. The **International Phonetic Alphabet** is the cross-linguistic standard, first

developed in 1888 and still evolving. The **ARPAbet** is an ASCII-only scheme covering American English, and it remains ubiquitous in engineering pipelines because it survives contact with legacy tooling that predates reliable Unicode support.

The course outline names three approaches to this conversion. They are best understood as a cascade of fallbacks rather than as competing designs.

Dictionary lookup

The most reliable method is to look the word up. A **pronunciation dictionary** maps written words to phone strings, generally with stress marked. The CMU Pronouncing Dictionary is the standard freely available resource for English; others, such as CELEX, add syllabification and morphological information. Entries look like this:

Word	ARPAbet with stress digits	Note
COUNTER	K AW1 N T ER0	1 = primary stress, 0 = unstressed
TABLE	T EY1 B AH0 L	the final vowel is reduced
RECORD (NOUN)	R EH1 K ER0 D	stress on the first syllable
RECORD (VERB)	R AH0 K AO1 R D	stress on the second syllable

The last two rows show why a dictionary alone is not enough. English has hundreds of homographs — words spelled identically but pronounced differently depending on their grammatical role. Something must determine the part of speech before or alongside the lookup, which means even the most classical synthesis pipeline needs a grammatical analyser.

Morphological decomposition

Dictionaries are finite; language is not. When a word is absent, the next resort is **morphology** — decomposing the word into parts that are known. A **morpheme** is the smallest meaning-bearing unit of a language; a **root** carries the central meaning and **affixes** modify it. If *tokenize* is in the dictionary but *tokenizers* is not, the system can analyse the unknown word as token + -ize + -er + -s and assemble the pronunciation from parts it already knows.

This is harder than simple concatenation, because adding an affix can move the stress and change the vowels: *PHOtograph, phoTOgrapher, photoGRAPHic*. The stressed syllable moves, and the vowels reduce and un-reduce accordingly. Handling this correctly requires knowing which affixes are stress-neutral and which are stress-shifting — a body of knowledge from generative phonology that the Week 6 reading list covers in more detail than is appropriate here.

Rule-based letter-to-sound

For anything still unresolved — personal and place names, borrowings, invented product names — the fallback is a set of ordered context-sensitive rewrite rules, each of the form “rewrite this letter as this phone when it appears between these contexts”. The rules are ordered and the first match wins, so the most specific must come first.

> **Worked example: ordered letter-to-sound rules for the letter c**
>
> **1.** c → [ʃ] before ia (special)
>
> **2.** ch → [k] in stems of Greek origin (chorus)

3. ch → [tʃ] (church)

4. c → [s] before e, i or y (cell, city)

5. c → [k] elsewhere (cougar)

Trace for cyclic: the first c precedes y, so rule 4 gives [s]; the second precedes l, so rule 5 gives [k]; the third is word-final, so rule 5 gives [k]. Result: [sıklık].

Now break it. Try Celtic, ocean, facade, ciabatta. Each requires either a new rule or a dictionary entry. This is the characteristic failure mode of rule-based systems, and the reason the modern fallback is a small neural model trained on the dictionary itself — an encoder–decoder with letters going in and phones coming out.

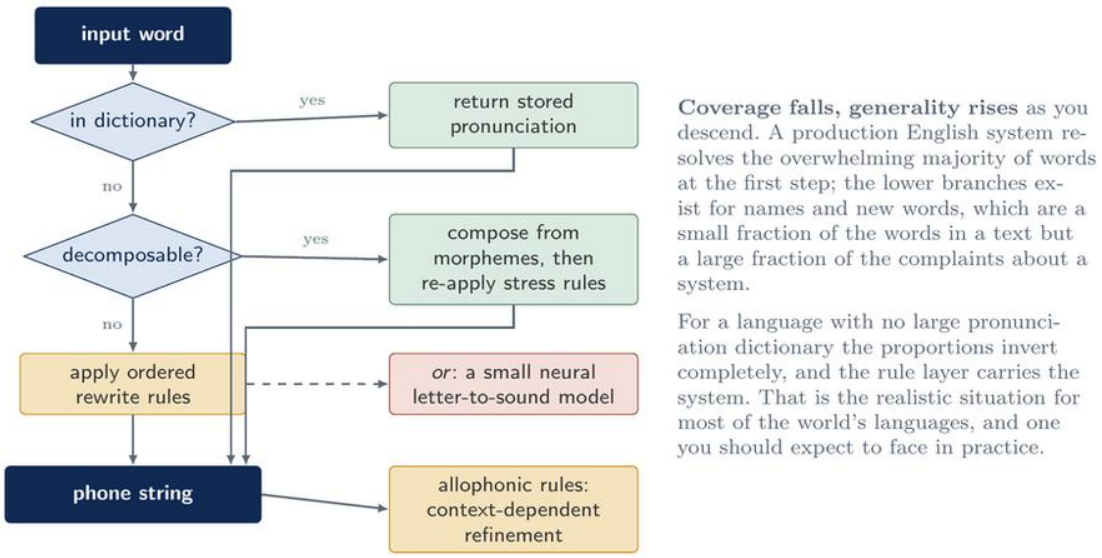


Figure 1.6. *The letter-to-sound cascade. The three approaches named in the course outline — dictionary, morphology, and rule-based conversion — are successive fallbacks, not competing designs. Coverage falls and generality rises as you descend.*

Allophones: one sound category, many realisations

The course outline lists allophones explicitly, and the distinction they encode is foundational to everything about speech in this course.

Phoneme and allophone

A phoneme is a contrastive sound category in a language: substituting one phoneme for another can change a word's identity. English /p/ and /b/ are distinct phonemes because pat and bat are different words.

An allophone is one of the physically distinct realisations of a single phoneme. The variants do not change word identity; which one occurs is determined by the surrounding context. English /p/ is aspirated — released with a puff of air — at the start of a stressed syllable, as in pin, but unaspirated after /s/, as in spin. Substituting one for the other yields an accent, not a different word.

Why it matters for synthesis: a system that ignores allophonic variation sounds foreign even when every phoneme is correct. **Why it matters for recognition:** the recogniser must map several acoustically dissimilar signals onto the same category, which is a large part of what makes the task hard.

Flapping is worth dwelling on, because it shows how tightly the parts of this subject are bound together. In many varieties of American English it neutralises the difference between writer and rider: the two words become acoustically near-identical. A recogniser therefore cannot recover the underlying consonant from the sound alone and must use its model of which words are likely in context. This is the tightest available illustration of why acoustic modelling and language modelling cannot be fully separated, and it will come back in Section 5.

Stress and intonation

The final synthesis topic in the course outline is **prosody**: the rhythmic and melodic properties of speech carried by pitch, duration and loudness rather than by the identity of the individual sounds. Prosody is the single largest contributor to whether synthetic speech sounds natural, and the aspect most often neglected by students building their first system.

> **Three levels of prominence**
>
> **Lexical stress** is a property of a word, listed in its dictionary entry. In surprised, the second syllable bears lexical stress. It is fixed and does not vary with context.
>
> **Pitch accent** is a property of an utterance. A speaker makes a word prominent by realising a pitch movement on it — but only on the syllable that already carries lexical stress. Which words receive a pitch accent depends on what the speaker is drawing attention to, and therefore on the discourse.
>
> **Vowel reduction** is the other end of the same scale: unstressed vowels weaken, often to the neutral vowel schwa, becoming shorter and more central. Not every unstressed vowel reduces.
>
> In short: the dictionary fixes where a word can be accented; the discourse decides whether it is; and everything unaccented tends to erode.

Above the level of the word, utterances have prosodic structure. They divide into intonation phrases, each carrying a characteristic pitch contour. A falling contour at the end of a phrase signals a statement; a rising contour signals, among other things, a question. The standard annotation scheme for English intonation is **ToBI**, short for Tones and Break Indices.

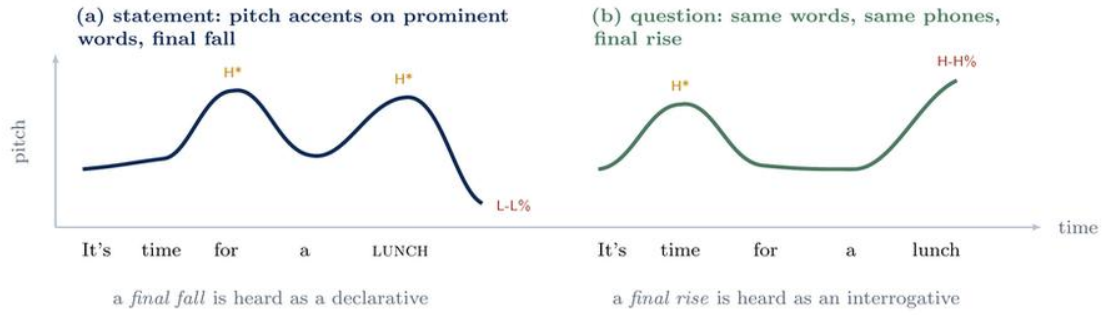


Figure 1.7. *Stress and intonation carry meaning that the sequence of phones does not. Both utterances contain identical words and identical phones; only the pitch contour differs, and it turns a statement into a question.*

Figure 1.7 makes the point that no amount of correct phone prediction substitutes for correct prosody. In a classical pipeline, predicting where phrase boundaries and pitch accents fall is an explicit prediction task trained on annotated speech. In a neural system it is not solved explicitly at all — the model learns it from data, which works well for languages with large recorded corpora and poorly for everything else.

Evaluating synthesis

Synthesis has no single correct output, so evaluating it is intrinsically harder than evaluating recognition. The reference method remains human judgement, usually reported as a **mean opinion score**: listeners rate naturalness on a five-point scale and the ratings are averaged. Two automatic proxies are useful during development. Running the synthesised audio through a speech recogniser and measuring the error rate gives a cheap measure of intelligibility. Passing the output and a reference recording to a speaker verification system gives a score for how well the target voice was matched.

5 Speech Recognition

Speech recognition is the inverse of synthesis: map a waveform to the string of words that produced it. It is the second topic in the course outline and, for most students, the one with the clearest route to a working artefact by the end of the course. Week 7 is devoted to it.

Speech recognition predates the computer. The first machine that could be said to recognise speech was a toy of the 1920s: a celluloid dog called Radio Rex, held in its kennel by a spring that released when it detected acoustic energy near five hundred hertz. Since that is roughly the frequency of the first resonance of the vowel in “Rex”, the dog appeared to come when called. Modern expectations are higher, but the underlying observation — that the identity of a speech sound is carried by the distribution of energy across frequencies — is the same one our feature extraction is built on.

Why it is hard, made concrete

Four dimensions of variation predict, with reasonable accuracy, how well a recogniser will work before you build it. They are worth internalising, because they let you set expectations honestly at the start of a project rather than discovering them at the end.

Dimension	Easy end	Hard end
Vocabulary size	two words (yes / no); the ten digits	sixty thousand or more, open-ended
Who is addressed	a human dictating to a machine; read speech	two humans in spontaneous conversation
Channel and noise	quiet room, head-mounted microphone	distant microphone, street, moving car
Speaker	the variety the system was trained on	other varieties, children, L2 speakers

The second row reflects something worth knowing about human behaviour: people simplify their speech when talking to a machine, or when reading aloud with no audience present. They slow down and articulate more clearly. Recognition of two people talking to each other — transcribing a meeting, for instance — is substantially harder than recognition of the same people talking to a device.

The fourth row is the one that matters most for deployment in a multilingual setting. A system trained predominantly on one variety of English will not be merely slightly worse on another; it will be substantially worse, and the degradation will fall unevenly across your users. Section 7 returns to this.

The front end: from waveform to features

Almost no speech system consumes raw amplitude samples. The standard front end converts the waveform into a sequence of feature vectors, one every ten milliseconds, and this same front end is shared by recognition, by language identification, by speaker identification and by wake-word detection.

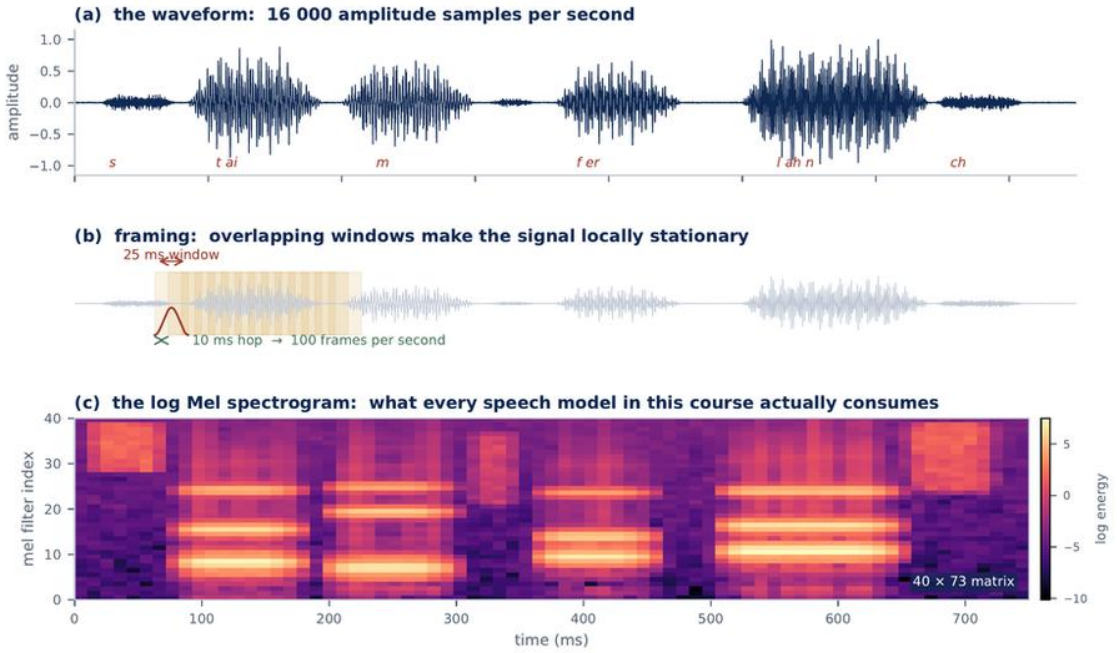


Figure 1.8. *Feature extraction, the shared front end of every speech system in this course. The waveform (a) is cut into overlapping windows (b), and each window is converted into a vector of log energies in perceptually spaced frequency bands (c). The result is a small matrix that a neural encoder can consume exactly as it would a sequence of word vectors. The signal shown is synthetic.*

Figure 1.8 shows the three steps. Three details recur constantly in practice and are worth stating explicitly:

- **The windows overlap.** A window of twenty-five milliseconds advanced by a ten-millisecond hop gives one hundred frames per second, and neighbouring frames share most of their samples. The overlap is what makes the resulting feature sequence smooth rather than jumpy.
- **The frequency scale is perceptual.** Human frequency discrimination is roughly linear below one kilohertz and roughly logarithmic above it. The mel scale spaces the analysis filters the way the ear allocates resolution, which means fewer numbers for the same useful information — a direct saving on an embedded device.
- **The logarithm matters.** Perceived loudness is roughly logarithmic in energy, and taking logarithms also compresses the enormous dynamic range of raw energy into a range a neural network can train on stably.

A further transform, producing what are called **mel frequency cepstral coefficients** (MFCCs), decorrelates these features. MFCCs dominated the field for decades and remain what you will find in classical toolkits and in most embedded speech libraries. Neural systems generally work directly from the log mel spectrogram, since a learned first layer can perform the decorrelation itself. You need to know both, because you will meet MFCCs whenever you touch legacy or resource-constrained code — which, in embedded work, is often.

The back end: two families

There are two ways to get from features to words, and choosing between them is the main architectural decision in any recognition project.

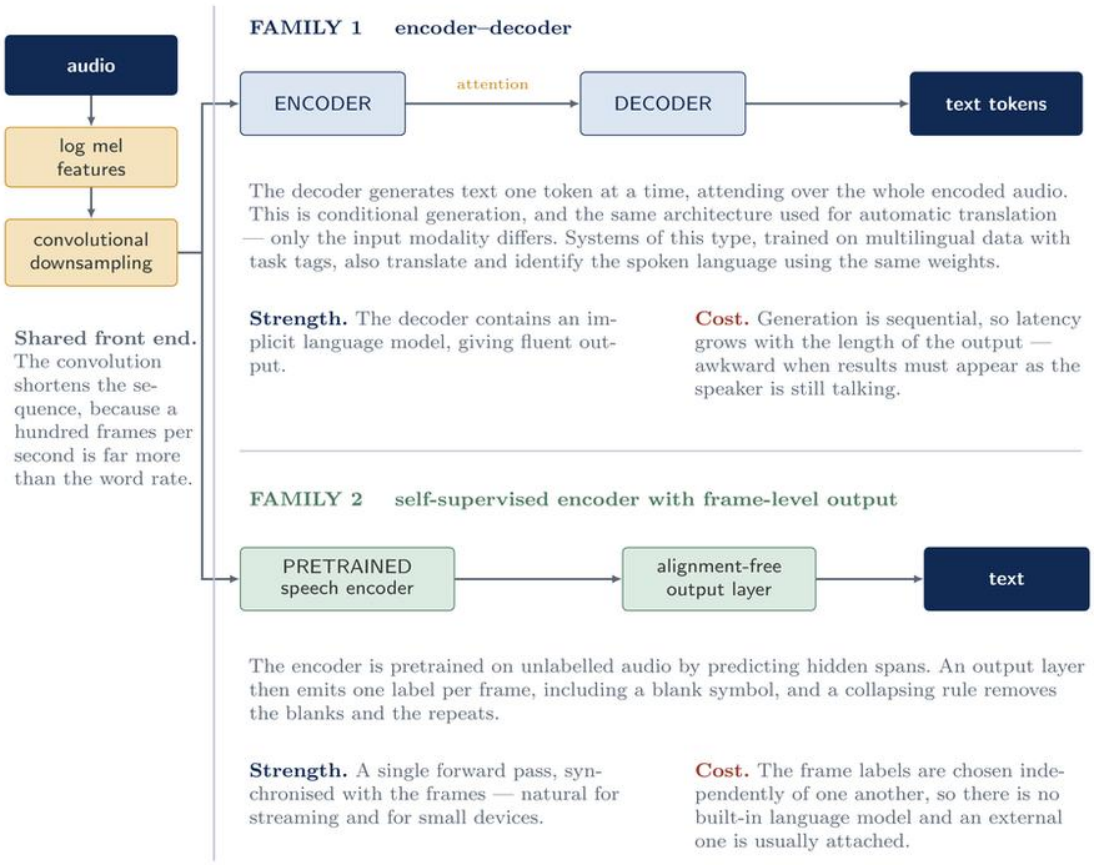


Figure 1.9. *The two families of modern speech recogniser. Both share the front end; they differ in how the encoded audio becomes text. The choice is driven by whether you need streaming output and a small footprint, or maximum accuracy on complete utterances.*

Figure 1.9 lays both out. Notice how little of it is new: encoders, decoders, attention and pretraining are the six ideas of Section 3, rearranged. What is genuinely specific to speech is the alignment problem. The audio arrives at one hundred frames per second; the transcript contains perhaps three words per second; and nothing tells us which frames correspond to which word. The second family solves this by allowing the network to emit a blank symbol at any frame and then summing the probability over every frame-level labelling that collapses to the same transcript. It is an elegant piece of dynamic programming and it is the reason that family exists at all.

Word error rate

Word error rate is the standard measure of recognition accuracy. It compares the recogniser's output to a reference transcript by finding the cheapest alignment between them — computed by the same **minimum edit distance** algorithm used for spelling correction in Section 6 — and counting the three kinds of error the alignment reveals: insertions, substitutions and deletions, divided by the number of words in the reference. The rate can exceed one hundred per cent, because there is no upper bound on the number of words a system may insert. It also weights every word equally, so a missed the costs as much as a missed not.

Worked example: computing word error rate

REF: the lecturer **marked** the **scripts** today

HYP: the lecturer **marketed** the — today **again**

The reference is six words long; there is one substitution (marked → marketed), one deletion (scripts), and one insertion (again). WER = 100 × (1 + 1 + 1) ÷ 6 = 50.0 %.

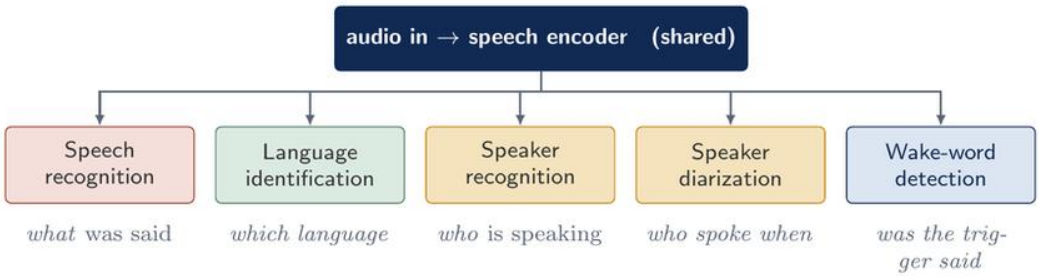
Note that marked becoming marketed counts as a single substitution, however similar the two words sound; and that the alignment itself was found by dynamic programming, not by reading from left to right.

6 Language Recognition

Language recognition is the third topic named in the course outline. The term is used in two ways in the literature, and this course covers both.

Spoken language identification takes an audio recording and determines which language is being spoken. It is essential for routing input to the right model in a multilingual product, for filtering and labelling training data, and for any service with an international user base. Architecturally it is a classification problem sitting on top of a speech encoder. Multilingual recognition systems often perform it as a by-product, since the language is one of the things they were trained to predict.

Written language identification takes a string of text and determines its language. It is one of the few places in this subject where a very simple method remains competitive: profiles of character sequences, built with the counting machinery of Week 2, classify most languages accurately on inputs longer than a sentence or two. Short inputs, code-switching within a sentence, and closely related languages all remain genuinely difficult. Practical tools for the laboratory sessions include the fastText language identification model and the langdetect library.



Speaker recognition divides into *verification* — a yes-or-no decision, is this speaker the person they claim to be, used for telephone banking and device unlock — and *identification*, matching a voice against a database of many speakers. **Diarization** answers "who spoke when" in a long multi-speaker recording. It typically detects segments containing speech, extracts a vector representing the speaker of each segment, and clusters those vectors. It is what makes an automatic meeting transcript readable. **Wake-word detection** is the one that must run continuously on the device. It uses the same front end as recognition followed by a small whole-word classifier, and it exists so that the rest of the pipeline — and the network connection — can stay asleep. It is the purest embedded problem in this course.

Figure 1.10. *The speech task family. All five tasks consume the same features and differ mainly in what sits on top of the encoder. Recognising this shared structure is what allows one deployed encoder to serve several product features at once.*

Figure 1.10 places language identification within the family of related tasks that appear in real products alongside recognition — speaker recognition, speaker diarization, and wake-word detection.

Wake-word detection is the one that must run continuously on the device; it uses the same front end as recognition followed by a small whole-word classifier, and it exists so that the rest of the pipeline, and the network connection, can stay asleep. It is the purest embedded problem in this course.

7 Translation, Summarizers, Spell Checkers and Grammar Checkers

The course outline groups four text technologies together. They are treated together in Week 5, and they are grouped here for a reason that will by now be familiar: three of the four are conditional generation, and the fourth is classification plus search. Once you have the toolkit of Section 3, none of them requires a new mechanism.

Automatic translation

Automatic translation is the best-developed of the four, and the one where the encoder–decoder architecture is seen at its clearest. It serves three distinct purposes in practice, and it is worth separating them because they impose different quality requirements. Information access — translating a page, a recipe, an official form — reduces the imbalance between what is available in widely spoken languages and what is available in others. Computer-aided translation produces a draft that a human translator corrects in a step known as post-editing; here the measure of success is how much time the human saves. Real-time communication between people with no shared language tolerates considerable error, because the participants can ask for clarification.

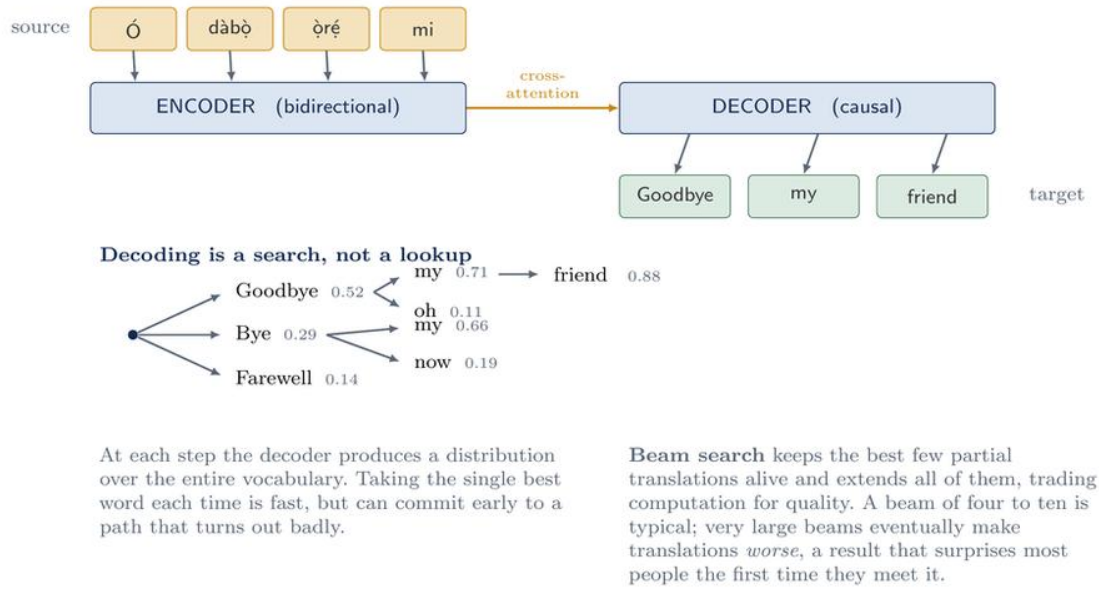


Figure 1.11. *Translation as encoder–decoder with beam search. The source is encoded bidirectionally; the target is generated left to right, each word attending back over the source. Substitute audio frames for source words and this becomes the first recogniser family of Figure 1.9.*

Figure 1.11 shows the architecture and the search. Two further topics matter for any deployment outside the small set of well-resourced languages. **Language divergences** explain in a structured way why translation is hard: languages differ systematically in word order, in which grammatical distinctions they mark at all, and in how they package meanings into words. **Low-resource**

translation covers the techniques — generating synthetic parallel data by translating backwards, transferring from related languages, exploiting text that exists in only one language — that make translation possible for pairs with little parallel text. This is the situation for the overwhelming majority of the world's roughly seven thousand languages, and the one you are most likely to face in practice.

Translation is scored by comparing a candidate against one or more human translations. **BLEU** counts overlapping word sequences; chrF does the same at the level of characters and is more robust for languages with rich morphology. Both are crude instruments, and both are increasingly supplemented by learned metrics and by human evaluation.

Summarizers

Summarization divides cleanly in two, and the division is worth knowing because it determines both the risk profile and the computational cost.

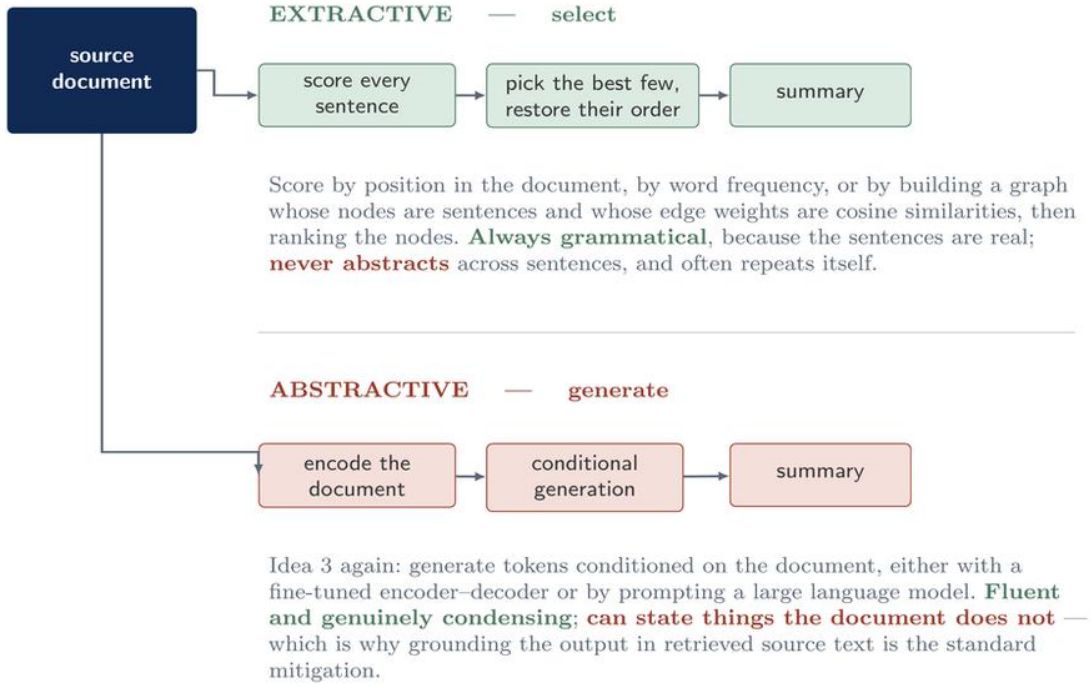


Figure 1.12. *The two approaches to summarization. Extractive methods are safe and cheap and remain the right choice on a constrained device; abstractive methods are better and riskier.*

Extractive methods select whole sentences from the document — scoring them by position, by word frequency, or by building a graph whose edges are cosine similarities and ranking the nodes. They are always grammatical, because the sentences are real, but they never abstract across sentences and often repeat themselves. **Abstractive** methods generate a summary as conditional generation over the document, with a fine-tuned encoder–decoder or by prompting a large model. They are fluent and genuinely condensing, and able to state things the document does not — which is why grounding the output in retrieved source text is the standard mitigation. Summaries are usually scored by ROUGE, supplemented by judgements of whether the summary is faithful to its source.

Spell checkers

Spelling correction rests on **minimum edit distance**: the smallest number of insertions, deletions and substitutions needed to transform one string into another, computed by dynamic programming. You will implement it in Week 2, and you will meet it again in Week 7 as the algorithm behind word error rate. One algorithm, two apparently unrelated applications.

A working checker needs three components beyond the distance function:

1. **Detection** — decide that a word is wrong. For non-word errors this is dictionary lookup. For real-word errors — their for there, form for from — lookup cannot help, because the written word exists; detecting these requires a language model, which is why real-word correction arrived decades later.
2. **Candidate generation** — propose plausible corrections, typically every dictionary word within an edit distance of one or two.
3. **Ranking** — choose among the candidates using the noisy channel model: pick the word that maximises a language model prior times an error model estimated from observed typing confusions. Weighting the edit operations lets the error model reflect that confusing m with n is likelier than m with q.

Grammar checkers

Grammatical error correction has two live approaches, and production systems usually combine them.

- **Rule-based.** A large hand-written pattern grammar over text that has been tagged for part of speech, each rule matching an error configuration and proposing a replacement. LanguageTool is the standard open implementation, with thousands of rules across dozens of languages. Rules are transparent, editable and explainable to the user — exactly what a writing aid needs — but laborious to author and blind to anything the author did not anticipate.
- **Neural.** Treat correction as translation from erroneous text into corrected text, using an encoder–decoder, or prompt a large language model. This handles unanticipated errors and awkward phrasing, but produces no explanation and will sometimes rewrite text that was already correct.

Grammar correction is evaluated with a precision-weighted F-measure rather than a balanced one, because a false correction annoys a user considerably more than a missed one does. That asymmetry — deciding which kind of error is worse, and choosing a metric that reflects the decision — is a general lesson, and it applies to every system you build in this course.

8 Mobile Speech and Language Applications

The final entry in the course outline is the development of mobile speech and language applications, and it is the one that distinguishes EAI 820 from a general course in language technology. It deserves a framework of its own.

The central tension is simple. Every technique described so far was developed under the assumption of a data centre: tens of gigabytes of parameters, a processor with very high memory bandwidth, and no particular concern for energy consumption. A phone, a microcontroller or a vehicle head unit has none of these. Every deployment decision follows from resolving that tension.

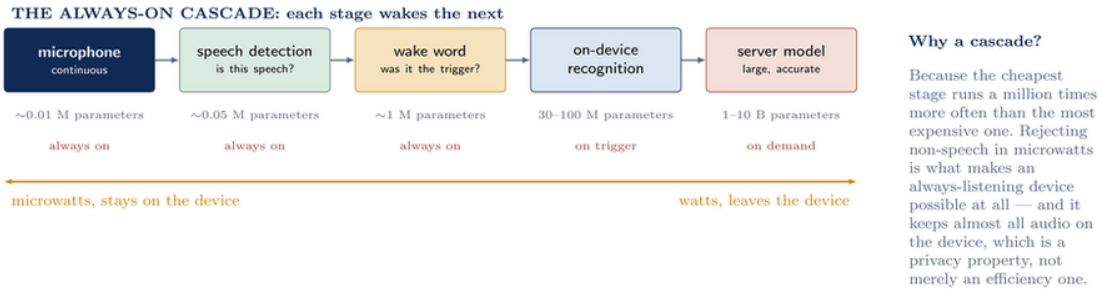


Figure 1.13. *The standard architecture for an always-on speech interface. Complexity, accuracy, power draw and privacy exposure all increase from left to right, while how often each stage is invoked falls by orders of magnitude at each step.*

Figure 1.13 shows the architecture that resolves it in practice: a cascade of increasingly expensive stages, each gating the next. This is why wake-word detection matters more to this course than its modest technical content suggests. It is the canonical embedded speech problem: it must be fast, tiny, and accurate enough that neither false wakes nor missed wakes annoy the user.

Four families of technique make a large model fit on a small device. All four are examinable, and Week 8 applies them to a pipeline you will have built yourself.

Technique	What it does	Typical effect
Quantization	Store and compute parameters as 8-bit integers, or fewer, not 32-bit floats	≈4× smaller, often faster; small accuracy loss
Pruning	Remove parameters, channels or whole attention heads that contribute little	2–10× fewer parameters; needs retraining
Distillation	Train a small model to reproduce the outputs of a large one	Much of the accuracy at a fraction of the size
Streaming	Restructure the model to emit output incrementally, not after the whole utterance	Latency becomes constant, not length-dependent

Notice how the architectural choice of Section 5 interacts with the last row. A recogniser built as an encoder–decoder must in principle see the whole utterance before it begins decoding; one that emits a label per frame streams naturally. For a device that must respond as the user is still speaking, that consideration can outweigh a point or two of word error rate. Architectural decisions taken for accuracy in Week 7 are re-examined for latency in Week 8, and some of them are reversed.

The practical toolchain follows the same logic: ONNX Runtime and TensorFlow Lite for quantised inference on device, Vosk for genuinely offline mobile recognition, and small distilled recognition models where a network connection can be assumed intermittently. Week 8 builds an end-to-end application — recognise, then translate or summarise, then speak — and then asks the harder question of what must change for it to run without a server at all.

9 Evaluation

A recurring difficulty is knowing which metric belongs to which task, so it is worth collecting them in one place. Two general points come first. Every metric below is a proxy for something we actually care about, and every one of them can be gamed. And no metric is interpretable without knowing the

test data: a word error rate of five per cent on read audiobooks and twenty-five per cent on spontaneous conversation may well come from the same system.

Task	Metric	What it measures	Week
Language modelling	Perplexity	Uncertainty on held-out text; lower is better	2
Classification	Precision, recall, F_1	Trade-off between false alarms and misses	3
Grammar correction	$F_{0.5}$	As above, weighting precision more heavily	5
Speech recognition	Word error rate	Edit distance to a reference transcript	7
Automatic translation	BLEU, chrF	Overlap with reference translations	5
Summarization	ROUGE	Overlap with reference summaries	5
Speech synthesis	Mean opinion score	Human naturalness ratings, five-point scale	6
Language identification	Accuracy, macro-F_1	Performance averaged fairly across languages	5, 7
Embedded systems	Latency, memory, energy	Whether it runs at all on the target device	8

Alongside the metrics you need the statistical apparatus that makes comparisons meaningful: holding out data properly, cross-validation, and significance testing to determine whether an observed improvement is distinguishable from noise. Reporting a single number from a single run is not an evaluation, and it will not be accepted as one in this course.

10 Data, Bias and Ethics

Systems trained on observed data reproduce the patterns in that data, including the ones we would rather they did not. Three consequences bear directly on the work you will do.

Recognition accuracy is not equally distributed. It varies systematically with dialect, with age, and with whether the speaker learned the language as a first or a later language. A system trained predominantly on one variety will underperform on others, and that underperformance is invisible to a developer who tests only on data drawn from the training distribution. The remedy is not primarily algorithmic: it is measuring error rates separately across speaker groups and treating the gap as a defect rather than a limitation. Any speech system built for assessment in this module should report error rates broken down by speaker group wherever the data permits it.

Generated text can be fluent and false. A model trained to predict plausible continuations has no mechanism for distinguishing plausible from true. The standard structural mitigation is to ground generation in retrieved source material rather than relying on what the model absorbed during training.

Voices can be cloned. Synthesis from three seconds of recorded audio makes impersonation trivial, and the consent, disclosure and detection questions this raises have moved quickly from research to law. They are part of the assessed material for Week 6, not an optional appendix to the technical content.

11 How the Course Is Organised

EAI 820 runs over eight weeks. Figure 1.14 shows the sequence, the course-outline topic each week addresses, and the laboratory work attached to it.



Figure 1.14. *The eight weeks of EAI 820. The deliberate split at week six — everything before it runs offline on any laptop, everything after it needs audio libraries and a graphics processor — exists so that problems with the computing environment do not consume the weeks in which the conceptual material is hardest.*

Why so much text before any speech? Because modern speech systems are built from the same components as text systems. A recogniser generates text conditioned on audio; a synthesiser generates audio conditioned on text. Attempting either in week two would mean treating it as a black box. Weeks 1 to 5 are what make weeks 6 to 8 comprehensible rather than merely operational.

Why no graphics processor until week six? Because a student debugging a driver installation is not learning about attention. Weeks 1 to 5 run in plain Python with NumPy on any machine, offline. The cost is that the audio work is compressed into three weeks; the benefit is that everyone arrives at those three weeks with working mental models rather than merely working environments.

12 A Short History

The history of this subject is worth knowing because its ideas cycle. Approaches abandoned as computationally infeasible return when hardware catches up; approaches abandoned as inaccurate return when data grows.

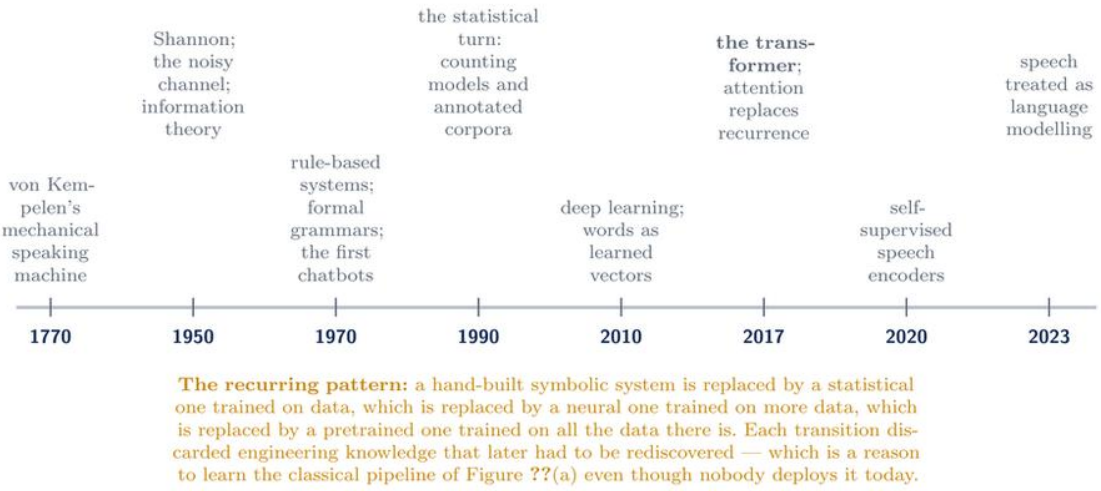


Figure 1.15. *Two and a half centuries of speech and language technology, compressed. A hand-built symbolic system is replaced by a statistical one trained on data, which is replaced by a neural one trained on more data, which is replaced by a pretrained one trained on all the data there is — each transition discarding engineering knowledge that later had to be rediscovered.*

Summary

- **Speech and language processing have converged.** Text and speech now share an architecture, a training strategy, and a formulation as generation over sequences of discrete tokens. That is why one course can cover both.
- **Language is hard for three nameable reasons:** it is ambiguous at every level, endlessly variable in its realisation, and productive, so no fixed list of words will ever be complete.
- **Six ideas underlie everything in this course:** discretise the input into tokens; represent tokens as vectors; predict the next token; learn by gradient descent; mix information across positions with attention; pretrain, then adapt.
- **Speech synthesis** has a classical pipeline — text normalization, letter-to-sound, prosody, waveform generation — whose vocabulary remains how the field describes failures, and a modern form as generation over discrete audio tokens.
- **The letter-to-sound problem is a cascade:** dictionary lookup first, morphological decomposition next, ordered rewrite rules or a small neural model last.
- **Prosody carries meaning that the phones do not.** Lexical stress, pitch accent and vowel reduction are three distinct things, and conflating them is a common and consequential error.
- **Speech recognition** shares a front end with everything else in speech, and splits into encoder–decoder systems and frame-synchronous ones. The choice is driven by streaming and footprint as much as by accuracy.
- **Language, speaker and wake-word recognition** are the same encoder with different heads, which is what makes one deployed model serve several product features.
- **Translation, summarization, spelling and grammar** are conditional generation or classification over the same machinery; minimum edit distance serves both spelling correction and the evaluation of recognition.

- **Mobile deployment is a cascade problem.** Cheap always-on stages gate expensive occasional ones, and quantization, pruning, distillation and streaming are what make the expensive stages fit.

Further Reading

The principal reference for this course is Jurafsky and Martin's *Speech and Language Processing*, third edition draft, which is freely available online and covers the foundational material, machine translation, phonetics, recognition and synthesis. Taylor's *Text-to-Speech Synthesis* remains the standard treatment of the classical synthesis pipeline, including letter-to-sound rules and prosodic modelling, and is the recommended companion for Week 6.

For practical work, the Hugging Face Audio Course is the recommended companion for Weeks 6 to 8, and the Hugging Face NLP Course for Weeks 1 to 5. Stanford's CS224N lectures are freely available and cover the foundational weeks at greater depth. For deployment, the documentation for ONNX Runtime, TensorFlow Lite and Vosk is more useful than any textbook, since that part of the field moves faster than books are revised.

Exercises

1.1 Consider the sentence *The old man the boats*. Identify the ambiguity and say which level of Figure 1.2 it belongs to. Would a speech recogniser or a grammatical analyser be more likely to get it wrong, and why?

1.2 For each of the following, state which configuration in Figure 1.4 it uses, and justify your answer in one sentence: (a) a spoken language identifier; (b) a system that produces a summary of a podcast; (c) a real-time speech-to-speech translator; (d) a wake-word detector.

1.3 Extend the letter-to-sound rule set of Section 4 so that it correctly handles *Celtic*, *ocean* and *facade*. Then find a word your extended set still gets wrong. What does this suggest about the long-run maintainability of rule-based conversion?

1.4 Give the ARPAbet transcription, with stress digits, for *synthesizer*, *allophone* and your own surname. Where your surname is not in a pronunciation dictionary, state which fallback in Figure 1.6 you used and what assumptions it required.

1.5 Explain, using the vocabulary of Section 4, why *writer* and *rider* may be acoustically identical in some varieties of English. What information must a recogniser use to tell them apart?

1.6 A reference transcript reads *please send the report to the dean before noon*. A recogniser outputs *please send report to the dean for noon*. Compute the word error rate, showing your alignment. Then argue, in three sentences, whether the rate overstates or understates the severity of these particular errors.

1.7 A window of 25 ms advanced by a 10 ms hop is applied to audio sampled at 16 kHz. How many samples are in each window? How many frames result from 3.5 seconds of audio? If each frame yields 80 coefficients stored as 32-bit floats, how much memory does the feature matrix occupy? Repeat for 8-bit quantised storage, and comment on the difference in the light of Section 8.

1.8 You are asked to build an offline voice-controlled irrigation controller for a farm with no reliable network connection, supporting about thirty commands in two languages. Using Figure 1.13, propose an architecture. State which stages run on the device, which recogniser family from Figure 1.9 you would choose and why, and what you would measure to decide whether it works.

1.9 Choose one entry from the course outline that this chapter treats only briefly. Find two recent peer-reviewed sources on it and write a one-page briefing note explaining how it relates to the six ideas of Section 3.

1.10 *Discussion.* Cloning a voice requires about three seconds of recorded speech. List three legitimate applications and three harmful ones. Propose one technical and one non-technical safeguard, and explain the limitation of each.