Episode Book

EPISODE #6379

Porting Voice Models to Core ML and MLX

PUBLISHED	31 October 2026 at 14:58 UTC
DURATION	30:46
LISTEN	myweirdprompts.com/episode/porting-voice-models-apple-silicon-80028/
HOSTS	Corn, Herman, Hilbert
SCRIPT MODEL	openai/gpt-6.1-sol
VOICES	chatterbox-regular (chatterbox-local)
PIPELINE	V5.26
SOURCES	20 cited by the research (listed at the end)
BOOK GENERATED	1 November 2026 at 07:41 UTC

SYNOPSIS

Porting speech and audio models from Linux PyTorch environments to Apple platforms involves distinct engineering trade-offs. Core ML requires graph capture and converter compatibility to leverage the Neural Engine, CPU, and GPU, packaging specifications and weights into structured bundles. Meanwhile, MLX requires direct reimplementation of the forward pass in array code, bypassing the Neural Engine entirely in favor of unified-memory CPU and GPU execution. Deploying voice models requires managing audio pipelines, memory limits, and hardware constraints.

TRANSCRIPT



Corn

Getting the weights onto the phone was the easy part. Getting the phone to agree about what to do with them... that's Daniel's prompt today. He has the porting problem in mind: a dictation or speaker identification model works on Linux or Android, and now somebody wants an iPhone app. Core ML and MLX appear on the Apple side, but what are you actually handing them, and how much of your working model survives the journey?



Herman

The checkpoint arrives. The computation needs separate negotiations.



Corn

Including the audio processing and the decoder, which are inconveniently involved in recognizing speech. Daniel wants the anatomy of the model package, the reality of PyTorch conversion, and the point where shipping ONNX Runtime or whisper.cpp becomes less work than persuading Apple's converter.



Herman

A successful Linux demo establishes that your model works in its original environment. It doesn't establish that another runtime can express every operation, preserve every shape assumption, or execute the same stopping rule.



Corn

The microphone has crossed the border. Its vocabulary hasn't.



Herman

Before we get inside the package, though, Core ML and MLX belong in different categories. That changes what a porting team is committing to.



Corn

Core ML gives me something I can put in an app. MLX sounds more like something I have to build the app around.



Herman

Core ML is Apple's on-device inference framework. Its tools turn a supported model into an artifact the framework can execute using the CPU, GPU, and potentially the Neural Engine. MLX is an array programming framework. You express the forward pass in its operations, load compatible weights, and execute that program.



Corn

Calling them two formats puts a document beside the program that wrote it.



Herman

With Core ML, you ask the converter to translate your captured graph. With MLX, you generally supply the model implementation yourself, or use an existing implementation of that architecture. Loading the checkpoint may be straightforward. Reproducing the computation is the substantive part.



Herman

Corn, if you were maintaining a small voice app, would you rather own that implementation or depend on conversion?



Corn

Conversion, until the converter becomes my largest unpaid colleague. I would rather preserve a model somebody already tested than write another version with fresh opportunities to transpose something. But if every update requires a special patch, eventually the supposedly automatic option becomes the more bespoke one.



Herman

That's a sensible dividing line. Core ML offers a system inference stack and access to hardware MLX doesn't currently use. MLX gives you direct control over the computation, with CPU and GPU execution. Neither choice removes the need to validate the port.



Corn

You installed the newest tool again, didn't you?



Herman

We're discussing categories.



Corn

That's what you call it before the environment breaks.



Herman

An isolated environment, Corn. I've acquired some wisdom.



Corn

You've acquired another environment.



Herman

Also true. The useful distinction here is responsibility. The Core ML converter owns much of the translation, until it encounters something it cannot translate. With MLX, the implementation is already your responsibility. You can make a cleaner engineering decision once you know which responsibility you're accepting.



Corn

The package anatomy matters because a checkpoint mostly gives me parameters. The deployed artifact needs enough structure to tell the runtime what those parameters mean.



Herman

A modern ML program commonly ships as an ML package, which Finder can present as one object although it is a directory. Inside are a manifest, a model specification encoded with Protocol Buffers, and separate weight data. The familiar names are Manifest dot JSON, model dot mlmodel, and weight dot bin.



Corn

A folder wearing a file extension.



Herman

The manifest describes the package's contents and format version. The specification describes inputs, outputs, model type, and computation. The weight data supplies the learned values used by that computation. They belong together, but they are physically separable.



Corn

That separation sounds useful for fine-tunes. Keep the structure, replace the parameters, avoid rebuilding everything upstream.



Herman

Potentially. A weight-swapping proof of concept demonstrates that reading the specification alone doesn't read the separate weight binary. But compatible structure matters. You cannot drop arbitrary tensors into the binary and expect the graph's references, shapes, and data types to remain valid.



Corn

Same sockets. Correct furniture.



Herman

Right sizes too. A model with the same architecture may still require exact agreement about tensor layout and serialization. The fact that two artifacts are separate makes controlled replacement possible. It doesn't make replacement self-validating.



Corn

I'd also want the labels. For speaker identification, the model could produce perfectly correct scores while the application attaches the wrong names. Nothing in a matrix multiplication knows that the second output is supposed to mean Herman.



Herman

The input and output descriptions are important for precisely that reason. The public specification also carries metadata such as author, provenance, and license. Application-specific interpretation may live outside it, though. A model package is not guaranteed to contain your complete product logic.



Herman

The specification has a version field and a choice of model kinds. Core ML supports more than neural networks: tree ensembles, support vector models, nearest-neighbor classifiers, regressors, and pipelines are represented too. A pipeline can connect several supported stages.



Corn

A voice developer sees a neural network and assumes the whole framework was built for that particular headache.



Herman

The modern ML program representation arrived with coremltools five. It provides typed execution and a GPU runtime backed by MPSGraph. The older neural network representation still exists, but it's the legacy route.



Corn

What does typed execution buy me beyond another way to reject my model?



Herman

Explicit numerical types and clearer execution semantics. That lets the compiler reason about supported operations and their precision rather than treating every intermediate as an unspecified blob. Then Xcode normally compiles the package into an mlmodelc bundle for deployment.



Herman

One distinction worth keeping straight... a third-party compiler implementation only supports ML program models. That limitation shouldn't be read as Apple having no ability to compile older neural network models. The particular compiler's supported formats and the system framework's supported formats are different things.



Corn

Good. Otherwise we'd have retired half the formats by footnote.



Herman

The compiled bundle is prepared for runtime execution. The portable specification and the compiled result serve different purposes. Shipping a checkpoint, shipping an ML package, and shipping the compiled app asset are different stages of the journey.



Corn

Does the package include the microphone processing? I suspect people picture a complete speech recognizer when the actual input description says a rectangular array of features.



Herman

It includes whatever supported computation you put into it. A package can accept an audio-derived tensor without knowing anything about microphone capture. Resampling, channel handling, feature extraction, tokenization, and text cleanup may remain application code.



Corn

Then the model's declared input is a contract worth reading before conversion. Raw samples versus mel features is a fairly consequential misunderstanding. It also determines whether a change in the app's audio frontend silently changes what the network receives.



Herman

Yes... and that is where comparing intermediate tensors becomes useful later. But stay with the anatomy for a moment. MLX supplies the machinery to express those operations as a running program. It has a C++ core, an MIT license, Python interfaces resembling NumPy, and neural network modules that feel familiar to PyTorch users.



Corn

Familiar enough to be dangerous.



Herman

Familiar enough to reduce the amount you must relearn. Not enough to guarantee identical behavior. There are also C, C++, and Swift interfaces. Computation is lazy, so constructing an expression doesn't necessarily execute it immediately. Work is evaluated when needed.



Corn

Finally, a framework with some restraint.



Herman

Its dynamic graph construction also accommodates changing shapes without the sort of slow recompilation users associate with some compiled systems. On Apple silicon, CPU and GPU operations use a shared memory model, avoiding separate copies merely to move an array between those devices.



Corn

Avoiding copies doesn't mean avoiding costs. The GPU can still be busy, the working set can still be enormous, and my dictation app can still make the phone unpleasant to hold.



Herman

Shared memory removes a class of transfer overhead. It doesn't create unlimited memory or free bandwidth. Nor does it give MLX a Neural Engine execution path. Its devices are currently CPU and GPU.



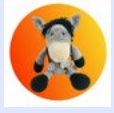
Herman

Would that missing Neural Engine path put you off, Corn, if the GPU version already felt instant?



Corn

Not on its own. I'd care whether it stays instant after prolonged use, whether it drains the battery, and whether it survives alongside other apps. A hardware badge wouldn't compensate for a recognizer that drops the end of my sentence.



Herman

Sustained behavior matters. MLX Swift has iPhone and Mac example applications, including chat and training examples, and the wider MLX examples include Whisper recognition. That establishes a real route for speech models. It doesn't turn an arbitrary



Herman (continued)

PyTorch checkpoint into an iOS application automatically.



Corn

There's a difference between an example launching and our app being ready to ship. Downloads, interruptions, memory pressure, microphone permissions... all still ours.



Herman

Some of the large-model iOS examples require the Increased Memory Limit entitlement. That isn't a blanket requirement for every small voice model. It illustrates that the application still lives under iOS memory constraints.



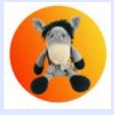
Herman

GPU work also needs a real device for this workflow. The iOS simulator doesn't emulate enough of the GPU to support those MLX examples. A successful simulator build cannot settle whether the model runs correctly on a phone.



Corn

A simulated phone declines to simulate the expensive part.



Herman

MLX also isn't restricted to Macs anymore. Linux CPU and CUDA installation options exist. It even has a GGUF reader. But file loading, execution backend, and architecture support are independent capabilities. Reading weights doesn't conjure the network that uses them.



Corn

With Core ML, we hand over a graph and depend on the converter covering its operations. With MLX, we re-express the forward pass and depend on getting that implementation right. We have a call on the package part.
Percy: Sorry, sorry, I have been nodding along to this show for a good while now, so this is



Corn (continued)

a bit embarrassing. What is a Core ML file, is it like a PDF you just open on the phone, or more like a recipe the phone follows?



Herman

It's... a recipe the phone follows, with the ingredients packed beside it, rather than a document you sit down and read.



Corn

You got there without saying protobuf.



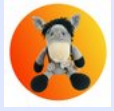
Herman

The next practical step for Core ML is graph capture. You take the PyTorch model in inference mode, supply representative inputs, and capture a graph using tracing or the newer export machinery. Then coremltools converts that captured representation into Core ML's representation.



Corn

Two chances to lose the original meaning before the phone sees anything.



Herman

Two distinct places to inspect, yes. Capture must represent the intended computation. Conversion must translate that representation correctly. Apple's documented guidance favors the older tracing path as the more established route. The export path was added in coremltools eight and is described there as beta.



Corn

Newer capture doesn't imply wider support downstream. The export graph may be cleaner for one compiler and contain operations another converter hasn't learned.



Herman

Apple's workflow guide gives roughly seventy percent PyTorch operation translation test coverage for that export path. That's a useful warning about coverage. It is not a seventy percent chance that your model converts, nor an exact census proving that every remaining operation is impossible.



Corn

My model only needs one missing operation.



Herman

A single unsupported operation can block a whole graph. Voice models deserve particular scrutiny around Fourier transforms, complex arithmetic, feature extraction, variable-length sequences, and decoding. Support depends on the exact operation form, capture route, converter version, and chosen deployment target.



Corn

Then nobody can honestly hand Daniel one permanent blacklist. Even a supported convolution might arrive surrounded by shape operations that the converter handles differently.



Herman

Correct. There are concrete examples, though. The coremltools tracker has an unsupported ATen alias operation. Another reported model works through tracing but fails through export at a narrow operation. Those aren't exotic speech algorithms. They're ordinary tensor plumbing encountered in particular graph forms.



Corn

The recognizer understood speech. The converter objected to slicing.



Herman

An unsupported alias doesn't mean the Neural Engine lacks the physical ability to preserve a value. It means that translation route doesn't know how to lower that captured operation. We should distinguish converter gaps from hardware limitations.



Corn

Could you remove those operations before capture, or are you stuck changing the converter?



Herman

Sometimes a decomposition or small model rewrite is sufficient. Other times you write a translation into existing MIL operations, Core ML's intermediate representation. That's the documented workaround for a missing conversion function.



Corn

Automatic conversion, with a compiler-writing interval.



Herman

You're implementing a small piece of compiler behavior, yes. It can be a sensible investment if the missing operation has a straightforward equivalent and the rest of the model works well. It becomes less attractive if you're accumulating special cases across every release.



Herman

Tracing introduces a different problem. It records the operations exercised by example inputs. Data-dependent Python control flow can get baked into the captured path. If your decoder stopped after a particular sequence, the trace doesn't necessarily express the general instruction to keep decoding until a stop token appears.



Corn

The example sentence becomes an architectural decision.



Herman

Potentially. Ordinary autoregressive speech decoding makes that consequential. You can often export a decoder step rather than the complete decoding loop, and let application code drive successive steps. That makes the boundary explicit.



Corn

Speaker identification could be easier there. Fixed audio window, embedding network, then a comparison or small classifier. Still preprocessing to match, but no text-generation loop waiting to surprise you.



Herman

Often easier, yes. The architecture matters more than the label voice model. A fixed-window speaker embedding model and a streaming transducer present very different conversion problems. Scripted PyTorch can represent control flow, but coremltools has limited support for scripted models, so that isn't a universal rescue.



Corn

I'd split off the frontend before rewriting a whole network. Keep audio decoding and feature extraction in ordinary code, convert the stable tensor network, and drive any decoder loop outside it. The danger is creating two subtly different definitions of the features.



Herman

Would you accept a slightly slower frontend to keep that boundary simple?



Corn

If it is actually a small share of the total time, yes. Not if we're moving the dominant cost onto one CPU core merely because it made conversion pass. I'd want a profile before congratulating myself.



Herman

That also affects partial delegation. ExecuTorch's Core ML partitioner can send supported subgraphs to Core ML while retaining other operations in ExecuTorch. One example explicitly excludes a matrix multiplication operation from delegation. Again, that illustrates a chosen partition, not a blanket claim that Core ML cannot multiply matrices.



Corn

The unsupported part stays home. Does all that traveling between runtimes erase the gain?



Herman

It can. Boundary crossings, synchronization, intermediate buffers, and scheduling all cost something. One large supported block may benefit substantially. A graph fragmented into many tiny islands can perform worse than a simpler single-runtime implementation.



Corn

You'd still have the maintenance burden of both runtimes. That's a lot to carry for a fast rectangle in the middle.



Herman

Precisely why whole-application measurements matter. Conversion success is only the first gate. Next comes numerical agreement. Reduced precision, different kernels, and changed execution order can produce drift even when every operation converts.



Corn

For dictation, I'd compare the text as well as tensors. A small logit change can select another token, and that changes the decoder's later inputs. For speaker recognition, I'd inspect scores near the decision threshold, not just examples with obvious matches.



Herman

Core ML's FAQ suggests CPU-only execution or full single precision as diagnostic comparisons when converted outputs show high error. Those settings help isolate whether precision or a particular execution backend contributes. They aren't necessarily the settings you ultimately ship.



Corn

Slow and correct earns a place in the investigation.



Herman

Corn's preferred deployment tier.



Corn

It has unusually good uptime.



Herman

Mine too, when you stop unplugging the experimental version.



Herman

The tooling has improved here. Coremltools eight point three added validators, model comparators, inspectors, benchmarking tools, and remote-device benchmarking. You can compare source and converted behavior and measure execution on the actual device rather than treating the final transcript as your only diagnostic.



Corn

Variable-length audio remains awkward even when the math agrees. A short utterance and a long recording shouldn't require pretending they have the same shape.



Herman

Flexible shapes are supported, but how you express them affects backend eligibility. Enumerated shapes let you specify a bounded collection of expected shapes, up to one hundred twenty-eight entries. Fully dynamic reshaping introduced during conversion can prevent Neural Engine execution for affected computation.



Corn

Successful conversion, wrong processor.



Herman

Or mixed execution. We shouldn't promise that enumerating shapes forces every operation onto the Neural Engine. Supported operations, shapes, and compiler decisions still govern placement. Padding or bucketing audio lengths can make the graph more predictable, at the cost of extra computation and application logic.



Corn

That cost can be worth paying, but it belongs in the benchmark. A network taking less time after we've padded twice as much silence is not obviously a better speech system.



Herman

ONNX Runtime becomes attractive when the model already exports cleanly to ONNX and its required operators are supported by the iOS build you plan to ship. You preserve an existing graph-based deployment route instead of maintaining another model implementation.



Corn

ONNX has its own operator versions and backend differences, though. I'd hate to escape conversion trouble by giving the same trouble a different extension.



Herman

It still needs validation. ONNX is the interchange representation; ONNX Runtime executes it. If you enable its Core ML execution provider, Core ML's delegation constraints reappear for that delegated portion. Shipping ONNX Runtime doesn't imply every operation suddenly reaches Apple's accelerator.



Corn

The runtime name doesn't tell you where the arithmetic happened.



Herman

Historically, ONNX was also a bridge into Core ML. The old onnx-coreml converter is frozen and unmaintained. Today a team may reasonably ship ONNX Runtime itself instead of using ONNX as an intermediate stop on the way to Apple's format.



Corn

GGUF seems like the opposite bargain. You accept a specific runtime's supported architectures rather than ask it to understand any graph you produce.



Herman

Yes, with an important naming correction. GGUF is closely associated with llama.cpp and supported model implementations. Whisper.cpp has historically used its own GGML-derived model formats. We shouldn't promise that a generic GGUF file is a whisper.cpp model.



Corn

Same neighborhood. Different door.



Herman

The shared engineering idea is architecture-specific inference code, often in C or C++, with defined weight layouts and supported quantization schemes. Whisper.cpp implements Whisper. It avoids needing a general PyTorch-to-Core-ML converter to understand that forward pass. It does not thereby run arbitrary speaker identification networks.



Corn

Nor does any quantization automatically work. The runtime needs kernels for the chosen representation, and the model still needs accuracy checks after quantization.



Herman

That scope is what makes the route powerful. If your architecture is already supported, a lot of implementation work has been done. If it isn't, you're considering a source-level port, not merely a new export command. MLX offers a related ownership bargain, expressed through its array framework rather than a custom C++ inference engine.



Corn

I'd stop pushing full Core ML conversion when the workaround code becomes harder to maintain than the alternative that already works. Especially if the alternative meets latency, memory, and battery requirements. But I'd keep trying if one isolated lowering unlocks a large, stable network.



Herman

That's where I land too. An existing Whisper implementation favors testing whisper.cpp. A clean ONNX deployment favors testing ONNX Runtime. An existing MLX architecture favors evaluating the Swift route. A compact, convertible network with sustained mobile use makes Core ML's hardware integration particularly attractive.



Corn

None of those choices means the failed converter attempt was wasted. You learn where the boundaries belong. What I wouldn't do is keep the converter alive purely because the first architecture diagram promised a single model box.



Herman

That diagram also hides preprocessing and decoding maintenance. Somebody owns those pieces even when they sit outside the model package. Coremltools nine adds newer deployment targets, including iOS twenty-six, and another Apple-side path, coreai-torch, now lowers ATen operations into Core AI operations.



Corn

Another supported-operations page to become emotionally attached to.



Herman

That newer path also brings host-platform constraints. The documented recent coreai-torch setup requires macOS on Apple silicon. I'd distinguish that from the broader coremltools workflow rather than announce that all conversion now requires a Mac. But it adds another environment for a Linux-based team to account for.



Corn

Your isolated environment needs an isolated computer.



Herman

Sometimes, yes. I can't defend the desk space. Even with a compiled artifact, though, the person who keeps the surrounding system running is usually the developer who...



Hilbert

Unjams the foot pedal, Herman. At the borough housing office it was Mrs. Eileen Mercer. The typists called her when the transcription pedal stuck, and the clerks called her when an address on a dictation tape didn't match the property card. She kept the actual work



Hilbert (continued)

moving. This was in seventy-four, when I was repairing office recorders. I was trying to get a repair invoice paid. That took longer than the repair.



Corn

An address disagreement can outlast any machine fault.



Hilbert

Six visits. Mrs. Mercer had to find the original requisition each time because accounts wanted the department name written another way. She typed a whole replacement invoice for me once. Carbon copies, no correction fluid. She could change the paper in her machine in eleven seconds. I timed her after she said I was holding the stack wrong. You gave that decoder loop a careful explanation earlier, Herman. Her loop involved walking upstairs with my invoice and coming down with a new objection.



Herman

I gave the developer too much credit for the surrounding work. Someone still has to reconcile what the recording says with what the office can actually accept.



Hilbert

Mrs. Mercer did that before lunch. After lunch she dealt with the tapes nobody had labelled. My mother still has the steel transcription desk against her garage wall. Too wide for the doorway into the house. I bought it when the office changed its furniture because Mrs. Mercer said the pedal bracket was sound. The transport cost more than the desk.



Corn

The bracket passed. The doorway wasn't consulted.



Hilbert

I wasn't putting a sound desk in the rain. Accounts finally paid after Mrs. Mercer crossed out Housing Maintenance and wrote Housing Repairs. Same room. Same man signing it. She carried it upstairs herself. Twenty-six pounds and forty pence.



Herman

The forward-looking question is who will maintain those boundaries as the tools change. We don't have a reliable head-to-head number for the same voice model across all these runtimes on the same iPhone. Core ML's device benchmarking tools can help a team answer it, but the result has to include the frontend, decoding, and sustained use, not just the attractive middle block.



Corn

For Daniel's porting problem, I'd choose the route that meets those requirements with a computation we can keep correct. Core ML buys integrated execution when conversion fits. MLX or C++ buys ownership of the forward pass. Neither buys freedom from the surrounding app.



Herman

Thanks to Hilbert Flumingtop for producing the show.



Corn

For more along these lines, there's episode three, Safetensors or something else; episode fifteen, AI Gets Personal; and episode two, Local STT For AMD GPU Owners. This has been My Weird Prompts, the human-AI collaboration podcast.



Herman

If you enjoy these deep dives, leave us a review wherever you listen. It helps other curious people find the show.



Corn

See you tomorrow.

SOURCES

What the research for this episode read before the script was written. Primary sources first.

1. **coremltools repo, releases 9.0 (2025-11-10), 9.0b1 (2025-07-28), 8.3 (2025-04-29); last push 2026-09-30. [primary]**
   https://github.com/apple/coremltools

2. **What Is Core ML Tools?, supported libraries table. [primary]**
   https://apple.github.io/coremltools/docs-guides/source/overview-coremltools.html

3. **MLModel Overview, Core ML specification contents. [primary]**
   https://apple.github.io/coremltools/docs-guides/source/mlmodel.html

4. **PyTorch Conversion Workflow, ~70% op coverage, trace vs export. [primary]**
   https://apple.github.io/coremltools/docs-guides/source/convert-pytorch-workflow.html

5. **coremltools FAQs, unsupported op handling, EnumeratedShapes/128, mlpackage and ML program history. [primary]**
   https://apple.github.io/coremltools/docs-guides/source/faqs.html

6. **Core ML protobuf spec, model types. [primary]**
   https://github.com/apple/coremltools/blob/main/mlmodel/format/Model.proto

7. **`.mlpackage` layout constants (model.mlmodel, weight.bin, weights/, Manifest.json). [primary]**
   https://github.com/apple/coremltools/blob/main/coremltools/models/utils.py

8. **Manifest.json / fileFormatVersion. [primary]**
   https://github.com/apple/coremltools/blob/37d01e77/modelpackage/src/ModelPackage.cpp

9. **MLX README, releases v0.32.4 (2026-10-14), v0.32.3 (2026-09-29), v0.32.2 (2026-08-25). [primary]**
   https://github.com/ml-explore/mlx

10. **MLX Swift iOS/macOS example apps. [primary]**
    https://github.com/ml-explore/mlx-swift-examples/

11. **Increased Memory Limit entitlement on iOS. [primary]**
    https://github.com/ml-explore/mlx-swift-examples/blob/main/Applications/LLMEval/README.md

12. **coreai-torch supported ATen ops. [primary]**
    https://apple.github.io/coreai-torch/main/api/supported-aten-ops.html

13. **mlprogram spec v5+ requirement for `.mlmodelc` compilation.**
    https://github.com/tmc/apple/blob/v0.6.12/x/coremlcompiler/compile.go

14. **weight.bin vs model.mlmodel separation.**
https://huggingface.co/Hironabe333/coreml-mlpackage-weightbin-content-swap-poc

15. **simulator GPU limitation, davidkoski comment 2024-08-10.**
https://github.com/ml-explore/mlx-swift-examples/issues/101

16. **`aten.alias.default` unsupported, opened 2025-05-17, updated 2026-02-26.**
https://github.com/apple/coremltools/issues/2517

17. **CoreML works with jit.trace but not torch.export (`narrow` op failure).**
https://github.com/pytorch/executorch/issues/9506

18. **CoreMLPartitioner, comment 2024-03-19.**
https://github.com/pytorch/executorch/issues/1268

19. **coreai-torch 0.4.1+ macOS/Apple-silicon requirement, no Linux/arm64 wheels.**
https://hbmartin.github.io/Foundation-Models-and-Core-AI-and-MLX-skills/part-08-coreai-pytorch-conversion/references/02-op-coverage-composites-and-externalization/

20. **HN Core ML Community Tools, 2017-10-05, 127 points.**
https://news.ycombinator.com/item?id=15406237