What eight gigabytes has to hold, *counted in bytes*

A phone's memory budget for a language model, worked through in bytes from Apple's documentation and published models, with the unknown lines left visibly blank.

MITWA LAB, MUMBAI

IN PLAIN WORDS

A phone's working memory is shared. The operating system, the apps waiting in the background and the app on screen all draw on the same 8 GB in an iPhone 16, and Apple does not say how much of it one app may take. A language model needs room for its weights, which have a fixed size, and for a running record of the text it has read and written, which grows with every word. This review works through that budget in bytes, using models whose sizes have been published and Apple's own documentation. It marks which figures were measured, which are our own arithmetic, and which nobody outside Apple knows. It ends with what the phone does when an app asks for more than there is, which is to close apps.

Eight gigabytes, before anything is taken away

Memory here means RAM, the fast working memory a running program reads from, as distinct from storage, which keeps files when nothing runs. Every iPhone 16 has 8 GB of RAM. The figure comes from a news report of an interview in which Johny Srouji, Apple's senior vice president for hardware technologies, confirmed it for all four models; the report calls it the first time Apple had publicly confirmed the iPhone 16's memory (Miller, 2024). We found no Apple document that states the figure.

We came to this subject from our technical note 03, which writes down the condition a model must meet to fit on a phone and observes that Apple does not publish the share of memory an app may use. Bloom is built for iPhone 16 and newer, and its shipped build takes 2.0 GB, 25% of 8 GB; how Bloom was compressed is outside the scope of this article, and every model named below belongs to someone else.

Apple describes RAM as a limited resource that apps, the kernel and the system's own processes all draw on, and warns that one app using too much can make the whole device worse to use (Apple, 2026e). How much the system and the person's other apps hold on an iPhone 16 is not published, and the nearest figures come from other hardware. Xiao et al. (2024) list 5.5 GB available of 8 GB on an 11-inch iPad Pro with Apple's M1 chip, and 4 GB of 8 GB on a Huawei Nova 7, and assume that devices with 8 to 16 GB keep at least 4 GB free once the operating system is counted. On a Xiaomi 14 Pro, one music app in the background took several hundred megabytes of that free memory.

A per-app limit that is not printed

Next comes a limit that iOS sets for each app, which depends on the device; an app that exceeds it is terminated (Apple, 2026e). None of the Apple pages we opened gives a value for any device. An app can ask how much room it has left, and the answer is its current limit minus its present footprint; Apple adds that the limit can change while the app runs and need not match the device's physical memory (Apple, 2026d). An entitlement lets an app tell the system that its core features may run better above the default limit, but the higher limit exists only on some device models, which the page does not name, and the app must still work without it (Apple, 2026a). In a 2018 developer talk, an Apple engineer called the limit fairly high, said it varies with the device, and said app extensions get a much lower one (Apple, 2018).

Apple's guide to termination reports works through a sample in which an app on screen was ended for crossing the per-process limit while holding 92,802 pages of 16,384 bytes, or 1.52 GB (Apple, 2026c). Its example report header comes from an iPad in 2019 running iOS 13.1.2, but the page does not say that the process entry belongs to the same report, and neither example says where the limit sits on an iPhone 16.

Liu et al. (2024), designing small models for phones, adopt the premise that an app should use no more than 10% of a phone's memory, citing a 2012 paper on datacenter memory; on an 8 GB phone that is 0.8 GB, by our arithmetic. It is a design choice, and the models discussed below with 3.8 to 8 billion weights take 1.8 to 4.2 GB. Xiao et al. (2024) left lower-tier Apple devices out of their study because strict per-app limits on iOS stopped those devices holding their test model, but neither they nor Liu and colleagues measured the limit on an iPhone.

Weights at sixteen bits and at four

A model's weights are the numbers it learned in training, and their size is their count times the bits stored per weight. Apple's machine learning research group published both sizes for one model, Llama 3.1 8B, which takes about 16 GB as 16-bit floating-point numbers and 4.2 GB as 4-bit integers (Apple, 2024). At 16 bits the weights alone are twice the iPhone 16's memory; at 4 bits they are just over half of it, before anything else is counted. Apple ran the 4-bit version on a Mac with an M1 Max and says it did not test output quality systematically, only on a few inputs.

Smaller models leave more room, and Microsoft reports that Phi-3-mini, with 3.8 billion weights at 4 bits, occupies about 1.8 GB and ran offline on an iPhone 14 at more than 12 tokens a second (Abdin et al., 2024). Xiao et al. (2024) measured a 4-bit Llama 2 model with 7 billion weights using about 3.8 GB on all six devices they ran on the processor and about 4.2 GB on the graphics processors, and they judge a 4-bit 7-billion-weight model the upper limit for most mobile devices.

Apple budgets its own model too, without printing the budget. Its on-device model of about 3 billion weights can be compressed to about 3.5 bits per weight without significant quality loss, Apple reports, and ships at 3.7 because that already meets its memory requirements (Gunter et al., 2024), which the report leaves unstated.

Pages the limit counts, and pages it may not

The weights need not count the way the rest of an app's memory does. iOS measures an app's use in pages, typically of 16 KB, and memory the app has allocated but not written to is clean and does not count, while memory it has written to is dirty and does (Apple, 2026e). The 2018 talk went further. Files mapped into memory from storage and only read are always clean pages, one engineer said, with machine-learning models among his examples, and the footprint the system judges an app by is its dirty plus its compressed memory (Apple, 2018).

It follows, by argument, that weights mapped read-only from a file might sit outside the per-app limit in a way that working state cannot. We have not found this established for a language-model file of several gigabytes on current iOS, and the talk is eight years old. Current documentation only partly matches its account. In 2018, iOS had no traditional swap and instead compressed pages left untouched; one current page adds that iOS may move a memory-hungry background app to solid-state storage (Apple, 2026e), while another still says iOS compresses pages that apps have not used recently (Apple, 2026f).

Even where mapping works, it has costs. Clean pages can be dropped and must then be read from storage again (Apple, 2018), and Alizadeh et al. (2024) describe flash as well short of RAM in both latency and throughput. Apple also lists thrashing the file cache, by reading and writing scattered parts of mapped files too often, as a reason the system may end a background app (Apple, 2026c); whether reading model weights can cause it is not documented.

The record that grows with every token

As a language model reads and writes, each attention layer computes a pair of vectors for every token, called keys and values, and keeps them so that later tokens need not recompute them (Apple, 2024). Weights are fixed; this stored state grows with the length of the input. For Llama 3.1 8B, Apple gives its size as 2 (keys and values) times 2 bytes per 16-bit number times 32 layers times 8 key-value heads times 128, times the number of tokens, about 1 GB at 8,192 tokens. By our arithmetic from that formula, each token adds 131,072 bytes, and 4,096 tokens hold about 0.54 GB. As an illustration only, 4.2 GB of weights plus 0.54 GB of stored state is about 4.7 GB, more than the 4 GB that Xiao et al. (2024) take as the

least an 8 to 16 GB device has free, before the app is counted.

Lu et al. (2024) ran 4-bit models of 0.1 to 3 billion weights on an Nvidia Jetson Orin board. With the maximum input length set to 2,048 tokens for every model, footprints ran from 275 MB to 2,456 MB and rose roughly in line with model size. In the paper's breakdown at each model's own maximum length, compute buffers and stored state took 85 to 90% of memory for the Qwen1.5 models at 32,768 tokens and 83 to 87% for the Qwen2 models at 131,072. Architecture moves these figures too; OpenELM-3B, whose attention heads share keys and values in groups, had 164 MB of stored state, 3.9 times less than StableLM-zephyr-3B. Apple's own on-device system model accepts at most 4,096 tokens a session, counting instructions, inputs and outputs (Apple, 2026b); the documentation gives no reason for the number.

The app around the model

The last subtraction is the app itself, since data structures in its executable and in the frameworks it links count toward its use (Apple, 2026e). In the 2018 talk's example, a JPEG of 590 KB on disk became 10 MB once decoded for display (Apple, 2018). App extensions have a much lower limit than foreground apps (Apple, 2026c). None of the sources we opened gives a figure for a typical app's own footprint.

LINE OF THE LEDGER	BYTES	STATUS
Physical memory, iPhone 16	8 GB	Reported from an interview
System, kernel and other apps	Not published	Unknown
Per-app limit	Not published	Varies by device and over time
Llama 3.1 8B weights, 16 and 4 bits	16 GB and 4.2 GB	Published, as model file sizes
Its keys and values, per token	131,072	Our arithmetic
The app around the model	Not published	Unknown

Two ways to keep weights off the ledger

The strongest case against this arithmetic is that a model need not sit in an app's memory at all. Alizadeh et al. (2024), at Apple, kept weights in flash and loaded only the parts each step needed, running models as large as twice the RAM on hand, with speed-ups of up to 4 times on a processor and 20 times on a graphics processor over naive loading. They tested on Macs and a desktop graphics card with about half of memory available for the model, and the method depends on most units in some layers being inactive for a given token. We found no shipping phone system built this way in what we opened.

The second route hands the model to the operating system. Android runs Gemini Nano in a system service, and Google tells developers that its downloads and updates then do not weigh on the app's storage or runtime memory budget (Google, 2026). Apple's Foundation Models framework also lets an app call the on-device model behind Apple Intelligence (Apple, 2026b), though the page we opened does not say whether that model's memory counts against the app. The bytes still exist in both cases; on Android they sit on the system's side of the ledger.

How closely the footprint follows the weights has been measured only in part. Xiao and colleagues found that memory use on mobile devices closely tracked the size of the weights for a 7-billion-weight model, and Lu and colleagues found footprint rising roughly in line with model size at a fixed 2,048-token limit, with exceptions for models with large vocabularies. Both groups used llama.cpp and neither ran on an iPhone 16. Apple's 4.2 GB is the size of a model file, which by itself says nothing about the memory the running model takes.

When the sum does not fit

iOS warns before it acts, on a best-effort basis. If demand rises faster than the warnings relieve it, the system goes straight to ending apps and records why, and an app that allocates a large block at once can crash before any warning arrives (Apple, 2026f). Nearing the limit produces a report that Apple describes as a warning, and exceeding it ends the app (Apple, 2026e).

When pressure persists across the device, the system terminates apps to take their memory back, an event Apple calls jetsam, and records a reason (Apple, 2026c). An app ended under the per-process limit had exceeded its own allowance; one ended for page shortage was in the background and lost its memory to the app in front. An app ended while on screen looks to the person as if it crashed (Apple, 2026c), and its next launch starts from the beginning, more slowly than a resume (Apple, 2026e). The cost need not stay with the app that caused it, because Apple notes that an app using all the memory available to it can lead the system to end other apps and system processes (Apple, 2026d). As an illustration, a model that stays

inside its own allowance could still end the music the person left playing.

What is not yet known

- What the default per-app limit is on an iPhone 16, and what it becomes with the increased-memory entitlement. Apple publishes neither, and we found no measurement on that phone.
- Whether current iOS leaves weights mapped read-only from a file out of an app's footprint, as the 2018 talk implies for clean pages, and how often such pages are dropped and read back under pressure.
- How much memory the system and background apps hold on an iPhone 16 in ordinary use, and how much that varies between people.
- Whether loading weights from flash on demand, shown on Macs and a desktop graphics card by Alizadeh et al. (2024), keeps a usable writing speed on a phone. The authors argue that it would carry over to phones with 4-bit weights, and leave the on-device implementation outside their paper.
- How the move of background apps to solid-state storage relates to the memory compressor that Apple's current pages still describe.
- Whether the 10% premise of Liu et al. (2024), which they support with a paper on datacenter memory, matches any current platform rule; no platform page we opened states one.

References

Abdin, M., Aneja, J., Awadalla, H., Awadallah, A., Awan, A. A., Bach, N., Bahree, A., Bakhtiari, A., et al. (2024). Phi-3 technical report: A highly capable language model locally on your phone. arXiv:2404.14219 (v4, 30 August 2024). arxiv.org/abs/2404.14219

Alizadeh, K., Mirzadeh, I., Belenko, D., Khatamifard, S. K., Cho, M., Del Mundo, C. C., Rastegari, M., and Farajtabar, M. (2024). LLM in a flash: Efficient large language model inference with limited memory. In Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (ACL 2024). arXiv:2312.11514. arxiv.org/abs/2312.11514

Apple (2018). iOS memory deep dive. WWDC 2018, session 416, video and transcript. developer.apple.com/videos/play/wwdc2018/416/

Apple (2024). On device Llama 3.1 with Core ML. Apple Machine Learning Research, published 1 November 2024. machinelearning.apple.com/research/core-ml-on-device-llama

Apple (2026a). com.apple.developer.kernel.increased-memory-limit. Apple Developer Documentation, Bundle Resources. Undated page; the year is the year we opened it, 12 November 2026. developer.apple.com/documentation/bundleresources/entitlements/com.apple.developer.kernel.increased-memory-limit

Apple (2026b). Generating content and performing tasks with Foundation Models. Apple Developer Documentation. Undated page; the year is the year we opened it, 12 November 2026. developer.apple.com/documentation/foundationmodels/generating-content-and-performing-tasks-with-foundation-models

Apple (2026c). Identifying high-memory use with jetsam event reports. Apple Developer Documentation, Xcode. Undated page; the year is the year we opened it, 12 November 2026. developer.apple.com/documentation/xcode/identifying-high-memory-use-with-jetsam-event-reports

Apple (2026d). os_proc_available_memory. Apple Developer Documentation, os framework. Undated page; the year is the year we opened it, 12 November 2026. developer.apple.com/documentation/os/os_proc_available_memory

Apple (2026e). Reducing your app's memory use. Apple Developer Documentation, Xcode. Undated page; the year is the year we opened it, 12 November 2026. developer.apple.com/documentation/xcode/reducing-your-app-s-memory-use

Apple (2026f). Responding to low-memory warnings. Apple Developer Documentation, Xcode. Undated page; the year is the year we opened it, 12 November 2026. developer.apple.com/documentation/xcode/responding-to-low-memory-warnings

Google (2026). Gemini Nano. Android Developers. Undated page; the year is the year we opened it, 12 November 2026. developer.android.com/ai/gemini-nano

Gunter, T., Wang, Z., Wang, C., Pang, R., Narayanan, A., Zhang, A., Zhang, B., Chen, C., et al. (2024). Apple Intelligence foundation language models. arXiv:2407.21075 (v2, 27 May 2026). arxiv.org/abs/2407.21075

Liu, Z., Zhao, C., Iandola, F., Lai, C., Tian, Y., Fedorov, I., Xiong, Y., Chang, E., Shi, Y., Krishnamoorthi, R., Lai, L., and Chandra, V. (2024). MobileLLM: Optimizing sub-billion parameter language models for on-device use cases. In Proceedings of the 41st International Conference on Machine Learning (ICML 2024), PMLR 235. arxiv.org/abs/2402.14905

Lu, Z., Li, X., Cai, D., Yi, R., Liu, F., Zhang, X., Lane, N. D., and Xu, M. (2024). Small language models: Survey, measurements, and insights. arXiv:2409.15790 (v3, 26 February 2025). arxiv.org/abs/2409.15790

Miller, C. (2024). Apple's Johny Srouji confirms all iPhone 16 models have 8GB RAM, more details. 9to5Mac, 13 September 2024. 9to5mac.com/2024/09/13/iphone-16-ram-apple-intelligence/

Xiao, J., Huang, Q., Chen, X., and Tian, C. (2024). Understanding large language models in your pockets: Performance study on COTS mobile devices. arXiv:2410.03613 (v5, 9 February 2026). arxiv.org/abs/2410.03613