A Guide to Local AI

Rok Sosic, rok@cs.stanford.edu

November 3, 2026

Preface

This document is a beginner's guide to local AI, which means running AI on your own computer. It is intended for computer-savvy readers who have used AI but are new to local AI and want to learn about it. It explains key concepts, components, and trade-offs that are involved in local AI. It is not a step-by-step guide on how to install and use specific tools. The guide began as personal notes when embarking on the local AI journey. The journey turned out to be far more complex than expected, so here is a document that shares the learnings. Hopefully, you will find them useful.

Overview

Local AI means running an AI model, and the software around it, on computers that you control rather than relying on a cloud AI, where the key components of the AI are executing on remote cloud infrastructure. A local AI system can still use the Internet or cloud services when you choose to enable them.

There are several reasons for local AI, some of which are the following:

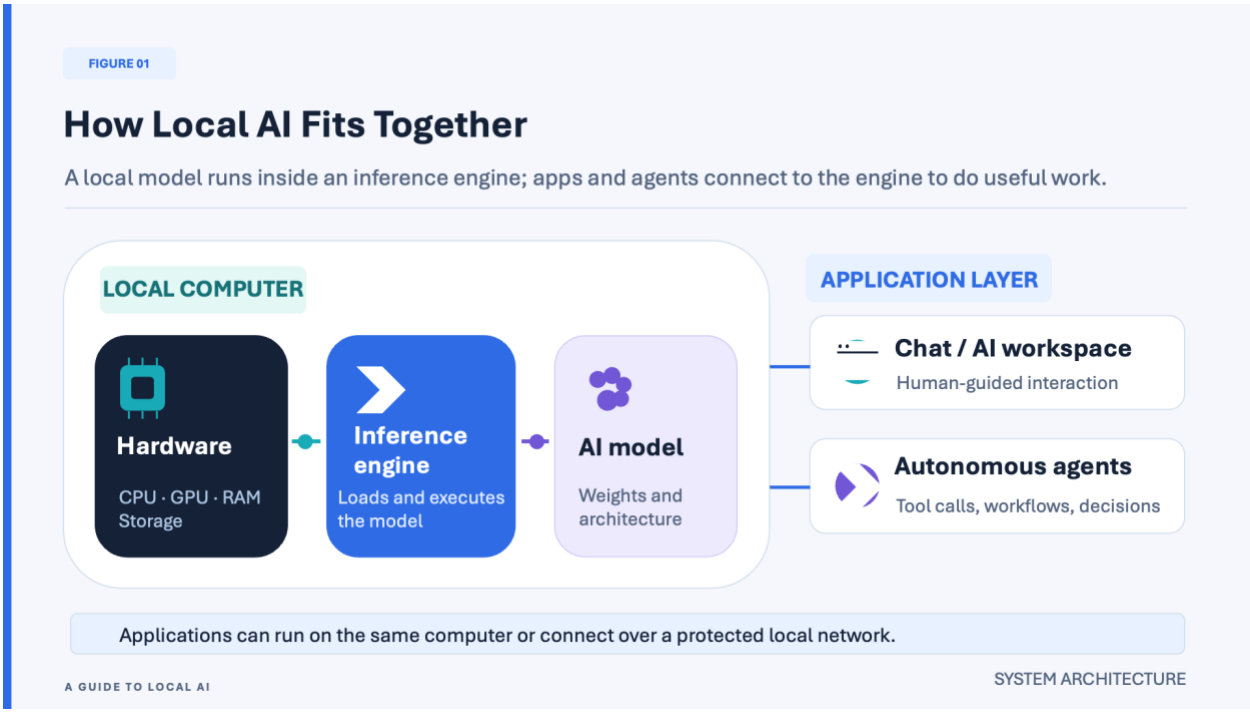
- You might not want to share sensitive private or business information with cloud AI providers;
- You do not want to have conversations about highly private matters with cloud AI;
- You want to control the components of your AI system. A cloud provider can change or even cancel the model at any time, or change the pricing or access policies, which might significantly impact your application;
- You want to build a significant knowledge base locally and not rely on a cloud provider. As someone quipped "Every time you ask a cloud model a question, its provider gets smarter and you get dumber."

Saving money is not the main reason for local AI. Prices for hardware skyrocketed over the last few years, driven by RAM prices increasing over 5x among other reasons. Cloud AI is priced reasonably and in general more capable than your local AI. Local AI will be cheaper than cloud AI in rare situations and might produce better results only for highly specialized, custom tasks.

Local AI consists of several components:

- computer hardware;
- an AI model;

- an inference engine;
- an agent harness.



A quick way to think about the components is that the AI model generates answers; the inference engine runs it; the workspace or agent harness manage conversations and actions. Tools give the harness the ability to take actions. Start with a local chat setup before enabling tools or autonomous agents.

Next sections describe these components in more detail.

Computer Hardware

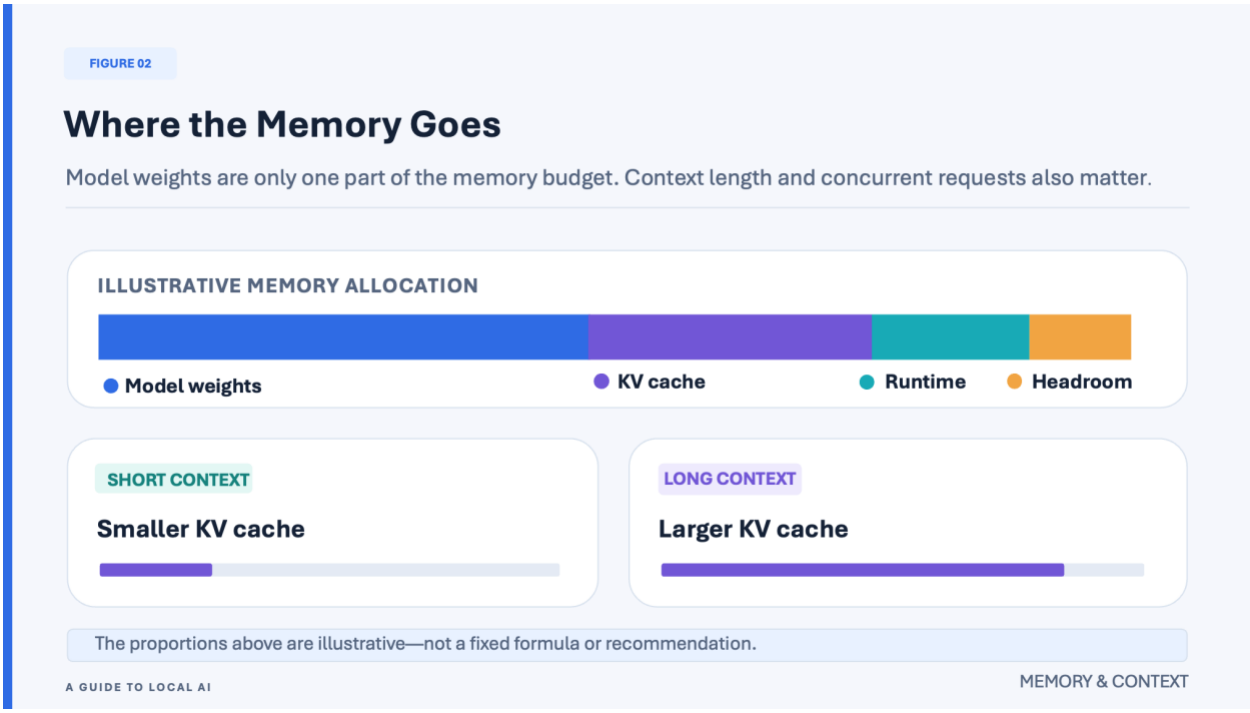
An essential component of your local AI is the computer that executes AI models.

Memory for AI. A key constraint is the memory (RAM) available for AI model weights, context, and other runtime data. More RAM lets you run larger models or use longer contexts. Larger models are often more capable, but size alone does not determine quality; model architecture, training, and the task also matter.

There is one caveat with RAM. Traditional, CPU-based RAM is not well suited for AI, which requires specialized RAM. That RAM comes in one of two forms. Either the RAM is so-called VRAM on a GPU card, or the computer system is designed with a unified RAM, which is shared between the CPU and the GPU units.

In case that you do not have specialized RAM, but only regular RAM and CPU, you are still able to run AI models, but they will be very slow with execution times 10x longer or more than systems with specialized RAM. Such a system could be used for experimentation with small models, but it will be most likely impractical for many common AI applications.

The size of this specialized RAM determines the size of the model that you will be able to use. As a quick rule of thumb, aim for the model to occupy at most 65% of the available specialized RAM with the rest reserved for other supporting AI data, such as a KV cache. On a GPU card, all the RAM is available for the model and its supporting data. In case of a unified RAM, the operating system and active applications will also reside in the same RAM, and only the reminder is available for AI, so you need to account for those. Keep available at least somewhere between 6GB and 16GB for the operating system and other applications, depending on the RAM in your computer.



Memory Bandwidth. Besides its size, another important characteristic of the specialized RAM is its memory bandwidth. The bandwidth is roughly proportional to the speed at which an AI model generates a response. A system with twice the memory bandwidth will generate output about 2x faster.

Hardware Options. Here are some hardware options for local AI with their approximate prices as of October 2026:

- Apple Mac Mini, M6, $1300, 32GB RAM, 256GB SSD, 153GB/s memory bandwidth
- Apple Mac Mini, M5 Pro, $2700, 64GB RAM, 512GB SSD, 307GB/s
- AMD Ryzen AI Halo, $4,000, 128GB RAM, 2TB SSD, 256GB/s

- NVIDIA DGX Spark, $5,000, 64GB RAM, 4TB SSD, 275GB/s
- Apple Studio, M5 Max, $5,400, 128GB RAM, 1TB SSD, 460GB/s
- Microsoft Surface RTX Spark Dev Box, $6,000, 128GB RAM, 2TB SSD, 300GB/s
- NVIDIA DGX Spark, $7,000, 128GB RAM, 4TB SSD, 275GB/s
    - was $4,000 when introduced in October 2025
- Apple Studio, M5 Ultra, $11,000, 256GB RAM, 4TB SSD, 1.2TB/s
- NVIDIA DGX Spark 2x, $14,000, 256GB RAM, 8TB SSD, 550GB/s
- Apple Studio, M5 Ultra, $22,500, 512GB RAM, 1TB SSD, 1.2TB/s

All the systems above have unified RAM. Not included are GPU-based systems, systems with less than 32GB RAM, or laptops. Although some laptops can run local AI, they might not be the best choice, since they are in general not meant to be running continuously. There are, of course, much more capable systems than those listed. A personal workstation that is optimized for AI can cost around $100k. A system that can efficiently run the most capable AI models, which have over 1T parameters, can easily cost $1m or more.

Hardware Selection. Here are some recommendations on how to select your system. For your given budget, try to maximize the size of the specialized RAM and the memory bandwidth. AMD offers the best price performance. Choose AMD, if the price is your main concern or if you want to have an x86_64 based system, running Linux or Windows. Apple offers a broad range of systems. Choose Apple, if the price is not your main concern, you want to have a clear upgrade path, or if you want to have a macOS environment. NVIDIA powers most of the cloud AI deployments. Choose NVIDIA or Microsoft, if you want an NVIDIA-based environment and want to develop for large scale NVIDIA installations.

Hardware Prices. Hardware prices and availability change quickly. Current expectations are that the shortage of RAM and thus outrageously high prices will last at least until the middle of 2028. If you want to use local AI in the near future, it might be the best to endure the pain and buy the hardware that you can afford as soon as you can. As of October 2026, the prices continue to rise.

AI Model

AI models are the brain of your local AI and are a key determinant for the capabilities of your system. An AI model is a set of files, which provide a description of the model's neural network. By far the largest part of the data are the weights for the neurons in the network, which are also called parameters. In general, the more neurons, i.e. parameters, are in the network, the more capable the network is.

Model Parameters and Representation. The number of model parameters is one key value that determines the size of the specialized RAM that the model requires on your computer. Another key value is the precision for the value of each parameter. The total required RAM size for the model is approximately the number of parameters times the number of bits for each parameter. A common size for the parameter values is 16 bits, or 2 bytes. A model with 8B

parameters, each represented by 2 bytes, will thus require 16 GB of RAM. Do not forget to take additional data, such as KV cache, into account as well, which is on top of the model RAM requirements. KV cache is discussed later.

For example, a popular model is Qwen3.8-27B, which uses a BF16 data representation. 27B indicates the number of parameters and BF16 indicates that the model uses 16 bits for parameter values. The model thus requires about 54GB of specialized RAM. It is expected to fit comfortably in computers with 128GB of unified RAM and can be probably used with 96GB RAM as well under carefully selected settings or even fit in 64GB RAM under very strict limits on the context size and other applications on the computer.

Model Quantization. The above calculation shows that even 27B models require expensive hardware at $4k or more when used at full precision. Fortunately, the number of bits per parameter can be reduced significantly without a major drop in model performance. This reduction in the number of bits per parameter is called model quantization, and is usually indicated with suffixes, such as Q8_0, Q5_K_M, and Q4_K_M, providing the number of bits as 8, 5, and 4, respectively. A Q4 variant of the model requires only 25% of the RAM for the full model using BF16. A Q4 variant of Qwen3.8-27B can fit in about 14GB rather than 54GB for the original model, which is a significantly cheaper option.

Although tools provide an easy way to create your own quantized model versions, there is no guarantee that the resulting model will provide high-quality results. It is better to use a quantized model version from a trusted source rather than creating your own model quantizations.

In general, for the same RAM size, a well-quantized larger model provides better answers than a smaller model at higher precision. More parameters with less precision are preferred in most cases over fewer parameters with higher precision. However, each bit reduction in parameter precision reduces the model performance as well. The performance starts to drop rapidly below Q4, so be careful with quantizations Q3 or less.

Dense and MoE Models. Another important model characteristic is the number of activated neurons for each generated token, a token corresponding roughly to one word of output. Most models, such as Qwen3.8-27B, are dense models, which means that all neurons are activated for each output token. However, some models support a technique, called Mixture of Experts (MoE), which activates only a subset of neurons for each output token. As a result, the MoE models need significantly less computation and can generate tokens much faster than a dense model with a comparable number of parameters. MoE models are thus well suited for local AI. An example of an MoE model is Qwen3.6-35B-A3B, where 35B denotes the number of parameters and A3B denotes that 3B of those are activated for each token. Although MoE models activate only a subset of neurons for each token, unfortunately, the entire model still needs to fit in the RAM. These models do not reduce the RAM requirements; they only speed up output generation. An exception are GPU-based systems, where the model can reside in the CPU RAM rather than entirely in the GPU VRAM. For MoE models, GPUs can run models that are much larger than their VRAM capacity.

Model Context. Models maintain their context, which is stored in the KV cache. The model context holds the information about input and output tokens, each token occupying one slot in the context. Each model specifies the maximum size of the context that it supports. The combined number of input and output tokens can thus not exceed the context size. The largest models today provide context sizes of up to 1M. Even medium-size models can provide context sizes of 256K. A larger context size supports larger prompts with more text or other information. Large prompts are needed for long running chat sessions or agents. The entire history of the conversation plus additional details are usually passed to the model with each new prompt, which leads to long prompts. The KV cache that holds the context occupies a large portion of the RAM. Depending on the model, it can even require more RAM than the model weights. Fortunately, while the number of model weights cannot be reduced, this is not the case for the context size. If the application does not require the entire supported context size, a reduced context size can be specified for the model, which can lead to significantly lower RAM requirements.

Repository of Public Models. The central repository for AI models is Hugging Face: https://huggingface.co/ which hosts millions of models. Essentially all publicly available open AI models are hosted on that site and can be downloaded for free. Of course, there are also closed AI models, most notably advanced models from OpenAI and Anthropic, which are not publicly available, and can only be accessed via cloud AI.

Another popular site for models is Unsloth, https://unsloth.ai/. Unsloth creates high-quality quantizations of the most popular models, so it is a good starting point to find a good model for your system.

Inference Engine

An AI model is just a bunch of parameter values holding weights for neurons in the neural network. A model cannot do anything on its own. Models require an inference engine, which loads a model in the memory, processes its input, and uses the model parameters to calculate the output. An inference engine and a model that the engine is executing must be located together on the same computer. More details about the inference engine operation are described next.

Model Loading. As the initial step, the inference engine loads an AI model from secondary storage to RAM. This step is performed only once for each model. After this step, the model is ready to be used and can serve many input prompts. The time for this step depends on the data transfer rate of your secondary storage. Since the AI models are many GBs in size, the loading time can be substantial. For example, a 100GB model takes over 8 minutes to load from a hard disk at 200MB/s transfer rate. On the other hand, it takes about 20 seconds to load from a PCIe based SSD at 5GB/s transfer rate. Whether the loading time is an issue will depend on your application and your workflow. If you work with many models that need to be changed often, it is

important to have fast storage with a short loading time. If you need to load the model only once and work with it for days or even weeks, then the load time does not matter.

Performing Inference. After the model is loaded in RAM, the inference engine performs the inference by repeatedly reading the prompts and generating output. Each prompt is input, most commonly text, from the user or an AI application. The inference has two main phases: input prompt processing (prefill) and output token generation (decode).

Input Processing. During prefill, the inference engine processes the input and computes the information that is required for token generation, such as maintaining KV cache. Input can be processed in parallel, so this task is in general compute bound. The rate of token processing during prefill can be important, depending on your application. If your main application is a chat with only a few lines of input text, the rate is less critical. However, if your model is used by agents, which send a large context with each prompt and which can easily exceed 100,000 tokens, a slow prefill processing can make the system unusable. For example, at a rate of 100 tokens/s, it takes over 15 minutes to process input with 100,000 tokens before a single token of output is generated. Modern hardware should be able to process input tokens at a much higher rate than 100/s. Nevertheless, prefill token rate is a critical metric that is often ignored, but needs to be considered. The rate can be improved by limiting the context size of the agent prompts or choosing a model with fewer parameters.

Output Generation. During decode, the inference engine uses the model parameters to generate output tokens, one token at the time. For dense models, each token requires activation of all the neurons in the network, or a subset of neurons for MoE models. The rate of output token generation is bound by the memory bandwidth of your hardware. The upper bound for the generation rate can be easily estimated from the memory bandwidth and the size of the model. For example, Apple Mac Mini, M6 has a memory bandwidth of 153GB/s. Assuming that the Q4 variant of Qwen3.8-27B requires 14GB of RAM, it should generate up to 153/14 or about 10 tokens/s. Similarly, for the Q4 variant of Qwen3.6-35B-A3B, assuming that each activation accesses 1.5GB of RAM (3B/2 for Q4), the generation rate goes up to 100 tokens/s. For an interactive chat, 10 tokens/s is an acceptable rate and most other applications might need even less than that. However, systems with many agents increase the need for faster token generation, since the number of prompts and corresponding results increase linearly with the number of agents. The rate of token generation can be improved by choosing a model with fewer parameters.

Inference Engine Packages. Some popular software packages that implement inference engines are:

- llama.cpp, it runs on a wide range of platforms, a good baseline choice;
    - https://github.com/ggml-org/llama.cpp
- MLX-LM, optimized for Apple Mac
    - https://github.com/ml-explore/mlx-lm
- vLLM, designed for GPU inference servers and large scale deployments
    - https://vllm.ai/

These packages are command line programs, which can be complex to use. If you prefer an easy-to-use application rather than dealing with command line programs, check out Unsloth Desktop: https://unsloth.ai/docs/desktop. It runs on all the platforms and integrates model management, execution, chat and other useful functionality in a single workspace. It uses llama.cpp internally as its default inference engine. However, the complexity of engine execution is hidden from the users. Unsloth Desktop is discussed more in the next section.

Model Service. An inference engine can provide a network-based model service. After the engine loads an AI model in the RAM, it opens a network port where it waits for prompts, performs an inference on each received prompt, and returns the result. An important aspect of being a network service is that it can be accessed from any computer on the same network. While this network access is convenient on your local private network, be careful on public networks. For example, if you are running a model service on your laptop on a network in a coffee shop and you are not careful with your settings, then any computer on the same network could access your service.

On the positive side, a model service on your local network at home or office that is behind a firewall can be accessed from any computer on the network. Network access allows you to use your local AI from one or more computers that are separate from the computer that is performing the inference. While the inference engine and the model must reside on the same computer, the applications that are using the model can be located on any computer on the same network.

Agent Harness

The term agent harness is used in different ways. Viewed broadly, an agent harness is any component outside of an AI model and its associated inference engine. In other words, an agent harness is any component that accesses the model service. The harness is essentially an AI workspace that provides chat and model access. Viewed narrowly, an agent harness is a system that supports autonomous agents working toward a goal with no or minimal human intervention. The harness manages an iterative workflow in which agents choose actions, call tools, and check results. Overall, the inference engine runs the model; the harness decides what information and capabilities to give it. For simplicity, the term agent harness is used in this document interchangeably for both views, the intended meaning should be clear from the context.

Connecting Harness and Model. The fact that a harness can access a model over the network leads to several options on how to combine them. We will use Unsloth Desktop to illustrate the options. Here are some possible configuration choices:

A. All components execute on one computer. Unsloth Desktop can be used to start the model on the same computer that it is running on, most likely your main laptop or desktop computer. This option is very convenient and probably the best choice to start with. However, all the components use the same RAM, so less RAM is available for the

model. Also, performing inference is resource intensive and might make the computer unusable for any other work.

B. The model service and Unsloth Desktop execute on two different computers on your private network. In this case, you need to have the model service already running on some computer on your network and you connect to it via the Unsloth Desktop on your personal computer. The Unsloth Desktop on your computer cannot start a model on another computer and that must be done with a separate earlier operation. The model service can be started via a command line on the second computer or by having on the computer a second, separate copy of Unsloth Desktop.
C. The model service is available over the Internet while Unsloth Desktop executes on your personal computer. In this configuration, you simply connect your Unsloth Desktop to the model service on the Internet. To be clear, this configuration is not local AI, since the model is running on the cloud, but it could be useful for situations where the power of the local model or local hardware is insufficient or where the use of cloud AI is preferred.

Using Cloud AI. With the configuration option C, many models are available over the Internet and each provider requires a separate registration and an associated payment. To avoid this complexity, a useful service is OpenRouter, https://openrouter.ai/, which provides a single interface with access to hundreds of models. You need to sign up only for their service and your Unsloth Desktop can access any of their 600+ models with a simple selection choice. Practically all models, either open or closed and including top models from OpenAI, Anthropic, Google and others, are available on the OpenRouter platform. Be aware that only a very limited number of models are available for free and that a vast majority of models require a usage-based payment, which is also integrated in the OpenRouter platform.

Unsloth Desktop. Besides running a model on a local computer or connecting to a remote model, Unsloth Desktop provides many useful features, such as chat, web search, tools, MCP, and integration with external agents, which allow users to perform a wide range of tasks. Unsloth Desktop is best described as a local AI workspace for running and interacting with models, with some agent-like capabilities built in.

Autonomous Agents. On the other hand, agent harnesses are optimized for automated agent operation, where agents act autonomously. These harnesses are designed to run an agent loop: the agents use the model to decide what to do next, call tools, inspect results, maintain task state, and continue iterating toward a goal. The harnesses allow you to create and manage many autonomous agents, each with a specific task and its own environment. The agents can collaborate with other agents and even create their own subagents for subtasks. Agents do not need to rely on the same model. Each agent can use a model that is specialized for its task and is different from models of other agents. A single agent could even use several models. A group of agents can utilize a mix of local and cloud models to perform their tasks. Of course, any models used must be available to the agents and their use supported by the harness. In general, agent harnesses rely heavily on persistent information and maintain long histories of agent activity.

An example of an agent harness is Agent Zero, https://www.agent-zero.ai/, which is one of the safest choices. Containerization is included by default and is a major part of its safety framework. Any access outside of its containers must be explicitly permitted. Other harnesses include Hermes Agent and DeepSeek Harness. Hermes Agent is the most permissive by default and any containers or other restrictions must be configured explicitly. DeepSeek Harness takes a middle approach and uses OS mechanisms to restrict agent actions on your local computer. Ultimately, all harnesses can provide a similar level of security, but it depends on the particular harness package whether the agents are constrained by default and they need to be given permissions to perform actions, or whether they are unconstrained and any restrictions needs to be configured. One approach to provide strong security for the agents is to run them on a dedicated virtual machine, which is separate from your local computer.

Applications

The sections so far only described the computing infrastructure that is required for local AI, but did not talk about the ultimate goal, which is using the local AI to perform tasks or execute applications. This section describes how local AI can be used for a wide range of applications.

Background. One way to look at the AI models is that they encode the vast knowledge that is available in the digital form and they provide a natural language interface for communication with the models. AI models can answer questions, generate text and code, and help analyze information through natural-language interactions. With a capable AI model, you have access to a countless number of experts in any domain you choose, and they are ready to converse with you or your agent as long as you want. It is up to you to figure out how to take advantage of this amazing capability.

Chat. The simplest way to start is to use chat. It is the premier application that initiated the current huge interest in AI. In its simplest form, the user, either a human or an agent, sends to the model some text as a prompt and gets back an answer. You can ask any question you want. For example, why is the sky blue, recommend me a healthy meal with eggs, or write me a poem about trees. In a multimodal chat, the prompt or the answer can contain other forms, such as images or even videos. A key point of chat is that it involves only the prompt and the model inference and that no other sources of information are used. Harnesses on the user side and the context on the model side maintain chat history, so the model can carry out a conversation that spans many prompts from the user. However, the model imposes limitations on the types of answers it can provide. Its capabilities do not extend beyond the knowledge that was used to train the model or the date when the model was trained. For example, it is unlikely that the model will be able to answer questions about sports results during the last week.

Web Search. To address model limitations, harnesses, such as Unsloth Desktop and Agent Zero, make it simple to combine chat with web search. If web search is enabled, they will iteratively call the model and perform a search on the web until the answer is ready. This approach effectively combines the knowledge encoded in the model and the knowledge on the

Internet, which removes model limitations. It can provide answers to the questions about the most recent events.

Tools. Chat and web search enable conversations, but they do not allow us to perform any tasks. For this purpose, harnesses provide tools, which is a mechanism to perform tasks on the local computer. Tools are usually programs or pieces of code, such as a Python program, that can be executed on the computer that is running the harness. For example, if the task is to summarize a document on our computer, then we can configure a tool that reads documents. The harness can use the tool to access the document and asks the model to summarize it. Tools are also widely used by the agents. Note that tools are local to the system with the harness or the agents.

Skills. While tools implement a specific function, skills are a similar, but higher-level harness mechanism for carrying out tasks. Skills are a collection of instructions that tell the harness or an agent how to accomplish a particular task by providing a description of workflows and resources. Skills can be viewed as a recipe for a task. For example, if we want to summarize a PDF document, a skill might contain the following steps: read a PDF document, perform OCR on its pages, and ask the model to summarize it. Of course, the tools to read the document and perform OCR must be available. Similarly to the tools, skills also need to be local to the system with the harness or the agents.

MCP Servers. Model Context Protocol (MCP) provides a standard way for AI applications to discover and use tools and other resources. An MCP server can run on the same computer as the harness or on another computer. These servers encapsulate the tools and resources on the local computer and make them available through a standard protocol. For example, if an MCP server supports a file reading tool, then the PDF summarization skill can be applied to the files on the computer with the server, even if the harness or an agent are on a different computer. Of course, the MCP server must be accessible over the network.

Custom Application. So far, we assumed that our harness is one of the commonly used packages. However, we can easily build our own harness, i.e. a custom program for our task. In fact, it might be the most sensible approach for certain applications, especially in commercial environments. For example, if the task is repeatable and performed often, it might make sense to create a specialized application for the task. Lets say that we collect many PDF documents every day and the task is to summarize the documents overnight and send a report in the morning. We already have a skill that reads PDFs via an MCP server, performs OCR and uses the model to summarize the documents. It is straightforward to extend the skill to incorporate the additional steps for the task. On the other hand, we could create an application that executes these steps without the need for the harness and the skills. Each approach presents a tradeoff; the skill gives more flexibility while the application offers more control and predictability. One approach is to start with the skills and transition over to a specialized application as the workflow becomes more understood.

Cloud AI. Although this guide is about local AI, cloud AI is a tremendous and indispensable resource that can be used to supercharge your local AI efforts. Top models are more capable and faster than the local AI in most cases and affordably priced. They are great at providing answers to general questions about your local AI, such as which model is the best for a particular task or a hardware configuration. Top models are also excellent at troubleshooting, such as "I encountered this error <error_text>. How do I fix it?" Furthermore, they are also very good at creating programs, such as the PDF summarization program discussed above. While such a program might require significant programming expertise and development resources in the past, it can be developed today by a top model using a prompt just a few lines long and with several follow-up short prompts requesting changes or additional functionality.

Security Considerations



Although the AI model runs on your local hardware, your information can still leak to the Internet. It is important to stay alert, especially if you are working with highly sensitive data. For example, as soon as you use features like web search, your harness will send queries to a web search service. The queries could contain information from your prompts. If you used your password or your social security in a previous prompt, those could be sent by the harness to the search service. The best way to avoid such risk is to not include any sensitive information in your prompt or any other resource that can be accessed by the harness.

Agents need even stronger safeguards. The agents act autonomously with even less control than is available in a harness conversation. They can make decisions and call tools without asking for approval. Give each agent access only to the folders, services, and credentials

needed for its task. Start with read-only access and require approval for actions that send data externally, modify important files, or run risky commands. Do not include any sensitive information in resources that are accessible to agents.

Tools, skills, and agents impose another danger, which is that they might perform a destructive operation, such as overwriting or deleting important files, either by mistake or in response to malicious content they encounter. Make sure to limit their environment and impose restrictions that prevent undesirable actions. A complementary mechanism is to have good backup and rollback capabilities to restore the environment, if needed.

Overall, AI offers powerful capabilities, and it can do amazing things, but it is not without risks. Be careful about what data it can access and what actions it can take. Keep private data from leaking out, limit what agents can change, and make sure important files can be recovered if something goes wrong.

Resources

- YouTube channels with information about hardware and models for local AI
  - Level1Techs, https://www.youtube.com/@Level1Techs
  - Alex Ziskind, https://www.youtube.com/@AZisk
- Inference engine
  - llama.cpp, https://llama-cpp.com/
- AI workspace and resource for quantized models
  - Unsloth, https://unsloth.ai/
- Repository of open weight models
  - Hugging Face, https://huggingface.co/
- Unified API for cloud models
  - OpenRouter, https://openrouter.ai/
- Agent harness
  - Agent Zero, https://www.agent-zero.ai/
- Benchmarks of AI models
  - Artificial Analysis, https://artificialanalysis.ai/
- Dashboard with an AI status
  - Artificial Acceleration, https://artificialacceleration.com/