# AgentFlux: A Framework for Privacy-Preserving On-Device Agentic Systems

We introduce a framework called AgentFlux for privacy-preserving, on-device AI agent workloads. By decoupling agentic tasks into function selection and argument generation, both tackled by local LLM orchestration, our system delivers accuracy approaching cloud-based models, while fully protecting user data from third-party exposure and enabling cost-efficient execution on consumer hardware. As part of AgentFlux, we provide the post-training pipeline and the inference pipeline that runs completely on-device using an example local base model. We instantiate the system on various use-cases ranging from financial applications, ai-browsers, and coding agents.

## Outline:

- AI systems are rapidly expanding from chatbots and media generation to robotics and financial applications.
- Leading AI platforms run in the cloud, sending all user queries that often include sensitive context (code, preferences, past interactions) to third-party providers.
- This creates two fundamental challenges:
    - **Data privacy:** User data, including medical and financial records, is routinely exposed to cloud providers.
    - **Cost and Latency:** Cloud APIs charge per token and throttle requests depending on the subscription packages the user pays for. Furthermore, it is estimated that all major providers are currently subsidizing costs, so the true steady-state cost for consumers is still unknown.
- The solution to both of the above problems lies in edge computing - manufacturers are rushing to put more powerful AI chips in laptop/desktop and mobile environments. Similarly, an array of open-source models has been released that can be run.
- However, running these models off the shelf on edge devices produces poor results in terms of both accuracy and performance. To address this, we introduce a new framework for edge computing that partitions workloads into two distinct tasks: selection of functions that need to be called, and argument generation. Partitioning the workload allows to create a hierarchical architecture that achieves end-to-end accuracy comparable with state of art cloud models with benefits of privacy and performance of consumer-grade GPUs.

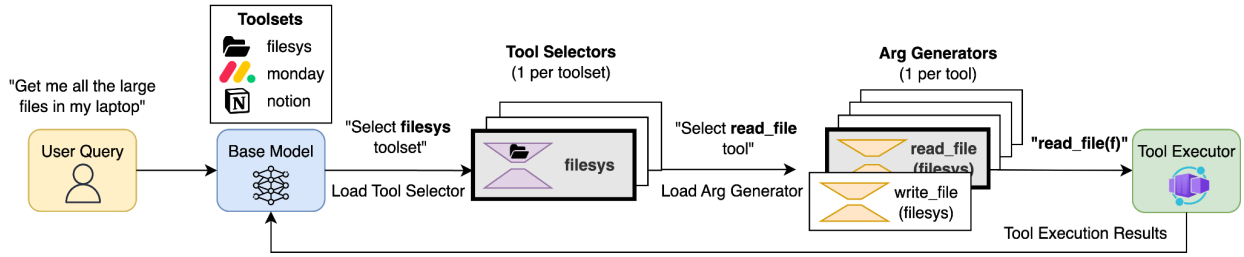
## Motivating use-cases:

- **Blockchains and financial applications:** Transactions are immutable and globally distributed, but wallet identities are (by default) pseudonymous. While the blockchain ledger is public, connecting transactions to real identities or generating regulatory reports (e.g., for taxes, GAAP compliance) requires combining sensitive off-chain data such as financial records, centralized banking activity, and KYC details. AgentFlux can enable local consolidation, analysis, and reporting across blockchain and traditional finance, keeping all private data on-device. Only anonymized outputs (never raw transactions or identities) need to leave the user's machine.
- **AI browsers:** AI browsers are similarly gaining popularity, replacing the traditional "click-and-tab" UX with prompt based interfaces, such as Comet, Dia, or Atlas. To generate the best results, these browsers need to (a) know the user's personal preferences, (b) history of interactions, and (c) have global access to the internet. While it's possible to instantiate the entire workload using a cloud service, this completely violates all user's privacy. AgentFlux enables local models to execute privacy-preserving tasks for sensitive data. AgentFlux can also be combined to work in collaboration with a cloud-based model, where the local model operates on private data and the cloud model operates on large public data (such as web search summarization).
- **Developer terminals and coding agents:** Coding and developer AI-systems are some of the most popular use-cases in AI. Terminals like Warp, Continue or Cursor allow developers to perform AI functions over their entire codebases. In many instances, these agents have access to all system files, while many workloads send all information in these files to third-party cloud services. For instance, common tasks include being able to set up dev environments, search through files and fix bugs, all resulting in total data leakage. LocalFlex can address these challenges by execution of parts locally and cost efficiently.

## Core Architecture:

Agentic systems autonomously solve complex tasks through iterative cycles: decomposing goals into discrete steps, executing each by invoking external tools, and dynamically adjusting based on tool outputs. Success hinges on LLM orchestration, which is the system's ability to accurately select the right tool and generate correct arguments at each decision point.

AgentFlux fundamentally reimagines this orchestration. Rather than relying on a monolithic LLM orchestrator, it employs multiple specialized LoRA adapters trained through a decoupled post-training pipeline and coordinated by a novel inference framework.




## Post-Training Pipeline

AgentFlux builds a post-training pipeline that automatically generates training datasets for all tools in a target application, then fine-tunes a local open-source LLM to produce two distinct LoRA adapter types:

1. **Tool Selector Adapter (per application)** – Functions as a classifier, identifying the optimal tool for each workflow step during inference.
2. **Argument Generator Adapter (per tool in an application)** – Produces precise, context-appropriate arguments for the selected tool at each step.

## Decoupled Inference Framework

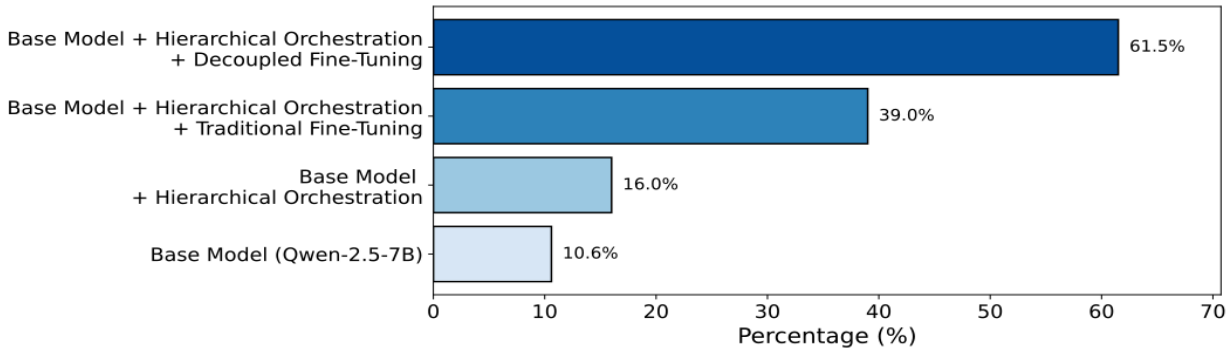
AgentFlux's inference engine restructures every agentic workflow step into two sequential sub-steps:

1. **Classification Sub-Step** – Dynamically loads the tool selector adapter to determine which tool to invoke.
2. **Argument Generation Sub-Step** – Dynamically loads the corresponding argument generator adapter to construct the tool's input parameters.

This modular, two-phase approach enables precise, adaptive orchestration while maintaining the efficiency and privacy benefits of local execution

# Key Results:

AgentFlux delivers significantly higher accuracy compared to off-the-shelf local LLMs and existing fine-tuning approaches, democratizing access to powerful and privacy-preserving agentic AI systems.




On a popular benchmark (MCP-Bench) that provides file manipulations for local and private files and directories, AgentFlux improves the accuracy of the base model from 10% to 62%, significantly closer to the accuracy of GPT-5-mini at 85%

AgentFlux also integrates effectively with a variety of applications such as Notion and monday.com, at times even outperforming larger local reasoning-based models (such as Qwen-3-32B)

---

**TL;DR:**
We present **DualTune**, a new framework that brings **frontier-level agent orchestration** to **local, on-device LLMs**. DualTune introduces **decoupled fine-tuning**, a novel post-training method that splits the tool-calling process into two specialized subtasks: **tool selection** and **argument generation**. By training separate LoRA adapters for each, DualTune dramatically improves local models' ability to call tools accurately, even within large toolsets, all while maintaining efficiency on consumer-grade GPUs.

**Paper:** https://arxiv.org/abs/2510.00229 (Code will be released soon)

## Agentic Systems

**Agentic systems** are AI systems that can autonomously complete **complex tasks** by **breaking them down into steps**, executing each step by interacting with external applications, and adapting their approach based on the output of the tools. The applications expose their functionality to the agentic systems via **API calls, or tools**, using standard protocols such as the **Model Context Protocol (MCP),** developed by Anthropic. Every MCP-compliant application defines a set of tools (called as a "**toolset**" or MCP Server), that contains all the tools belonging to the application, along with their descriptions and instructions on how the LLM can use them. The MCP protocol is used in various production systems such as VSCode, Cursor, Claude,

ChatGPT, and so on, and contains support for hundreds of applications such as Notion, monday.com, etc.

Agent systems contain a **Large Language Model (LLM) that acts as an orchestrator**, which takes as input a user task, along with a list of MCP applications that it can use for completing the task. The orchestrator then works through the task step-by-step: it calls a tool, stores the results, uses those results to decide the next action, calls another tool, and builds up a collection of information from each step. The process continues until the orchestrator has gathered enough data to provide a complete response to the user's original request. The **effectiveness of an orchestrator** depends on its ability to **break down the task into multiple steps**, and at every step, **call the right tool** with the **right arguments**.

## Need For On-Device Agentic Systems

Performing agent orchestration using frontier LLMs via cloud APIs poses two major problems:

- **Privacy:** Exposing local data (e.g., files, project notes) to remote APIs is risky.
- **Cost & Latency:** Continuous orchestration calls to frontier models are slow and expensive. For example, <>.

Running LLMs locally promises privacy-preserving and low-latency execution. However, **local models often struggle** to perform tool selection and argument generation reliably.

## Analysis of On-Device Agentic Systems

**Benchmark used:** We run 50 tasks from the filesystem toolset, generated using the MCP-Bench benchmark
**Tool-calling accuracy (ToolFit):** We use GPT-5 as the judge, which is provided with the ground-truth state of the filesystem for each task, allowing for a deterministic and accurate assessment of the final outcome. We assess orchestration performance using **ToolFit**, a metric which evaluates the output of agent systems on a 0-10 scale for tool-calling proficiency. A model's score represents the proportion of expected tool outputs successfully generated relative to the expected tool calls

| Model | ToolFit (%) | Reasoning | Frontier |
|---|---|---|---|
| Qwen-2.5-7B | 16.0 | No | No |
| Qwen-3-8B* | 34.2 | No | No |
| Llama-3.1-8B | 42.4 | No | No |
| xLAM-2-8B | 15.8 | No | No |
| Qwen-3-8B | 52.3 | Yes | No |

| Model | ToolFit (%) | Reasoning | Frontier |
|---|---|---|---|
| GPT-5-mini | 88.5 | - | Yes |

Evaluations on the MCP-Bench benchmark reveal that local LLMs perform poorly as orchestrators:

- They **misidentify tools** when multiple similar tools are present.
- They **fail to generate valid arguments**, often producing syntactically incorrect or incomplete calls.
- Long tool descriptions **inflate context length**, degrading both accuracy and latency.

| Tool selection Model | Argument Generation Model | ToolFit (%) |
|---|---|---|
| Qwen-2.5-7B | Qwen-2.5-7B | 16.0 |
| Qwen-2.5-7B | GPT-5-mini | 28.8 |
| GPT-5-mini | Qwen-2.5-7B | 60.8 |
| GPT-5-mini | GPT-5-mini | 88.5 |

**Takeaways:** the **performance bottleneck is primarily for the tool selection** as changing the tool selector. Tool selection and argument generation fundamentally differ in their nature, and can benefit from different optimizations. **Tool selection** is a **classification problem**, which requires identifying the appropriate tool at every step in the agent workflow from the available options. **Argument generation**, on the other hand, requires generating the right arguments as well as producing a structured output with **syntactic accuracy**.

## The Core Idea: Decoupled Fine-Tuning

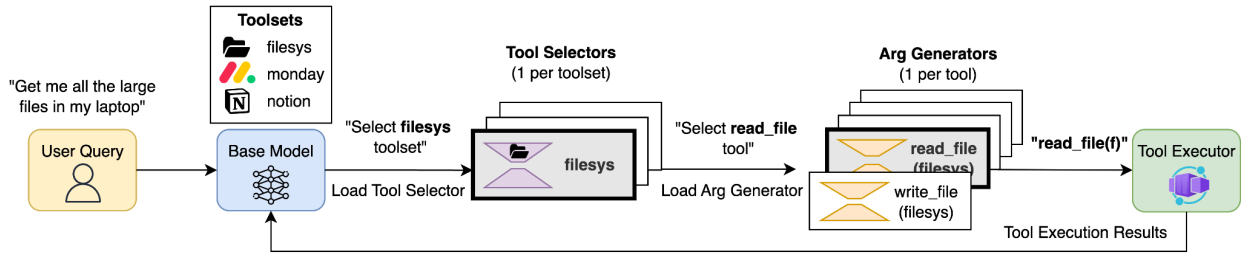
**DualTune** introduces **decoupled fine-tuning**, a two-stage post-training method that isolates the subtasks of tool calling:

1. **Tool Selection Adapter**
    - Trained as a **classifier** that identifies which tool to call next.
    - Uses loss masking to optimize only over the tool name tokens.
2. **Argument Generation Adapter**

- Trained individually for each tool to generate well-structured arguments.
- Uses masking over argument tokens only, ensuring syntactic precision.

Each adapter is a lightweight **LoRA module**, enabling fast swap-in/out at inference time.

DualTune's dataset pipeline automatically generates **synthetic tool-calling traces** using GPT-5-mini, producing balanced and diverse data across all tools.
The result is a scalable fine-tuning pipeline that can be retrained quickly when new tools are added.




## Hierarchical Orchestration for Scalability

To handle large toolsets without overwhelming the attention span of local models, DualTune employs **hierarchical orchestration**:

1. **Toolset Selection (High-Level Routing):**
   The base model decides which toolset (e.g., Filesystem, Notion, monday.com) is relevant.

2. **Tool Selection (Within Toolset):**
   A specialized LoRA adapter for that toolset selects the exact tool to invoke.

3. **Argument Generation:**
   The LoRA adapter for the chosen tool generates its arguments.

This hierarchy shortens context length, improves accuracy, and allows the system to scale to **dozens of tools** without slowing down inference.

## DualTune Inference Framework

DualTune integrates all components into a **Rust-based orchestration engine** that executes tools in **secure containers**.

It relies on **vLLM** for dynamic LoRA management, enabling DualTune to **load, unload, and swap adapters in real time** with negligible overhead.

The full inference loop:

1. **Toolset selection** by base model.
2. **Tool selection** by corresponding adapter.
3. **Argument generation** by the tool-specific adapter.
4. **Execution** inside a containerized sandbox.
5. **Observation and continuation** until the system outputs a “summarize” token.

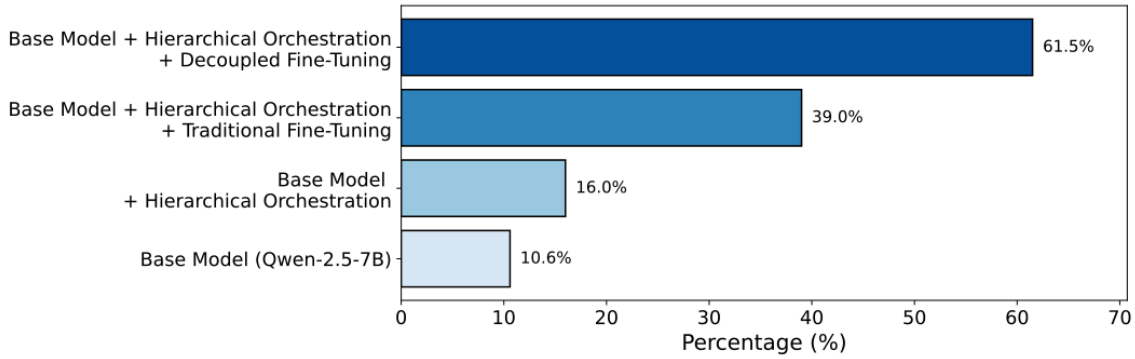
This modular pipeline lets local models handle complex multi-step tasks efficiently and securely on end-user hardware.

## How Well Does It Work?

**DualTuneModel-7B**, a Qwen-2.5-7B fine-tuned using decoupled fine-tuning, was evaluated on **MCP-Bench** and a custom **DualTune-TestSet** (covering Filesystem, Notion, and monday.com).

| **Model** | **Filesystem** | **Monday** | **Notion** | **Reasoning** |
|---|---|---|---|---|
| **DualTuneModel-7B** | **61.5%** | **43.2%** | **71.8%** | No |
| Qwen-2.5-7B | 15.0% | 19.2% | 33.4% | No |
| Qwen-3-32B-Quant | 58.6% | 37.0% | 85.6% | Yes |
| GPT-5-mini | 88.4% | 76.4% | 91.6% | - |

### Accuracy of Decoupled Fine-tuning v/s Traditional Fine-tuning:



### Highlights:

- DualTune improves **tool-calling accuracy by up to 60%** over the base model.
- It matches or exceeds reasoning models **2× its size**, but with lower latency.
- Decoupled fine-tuning doubles improvement over traditional fine-tuning on the same dataset.

## Why It Matters

DualTune bridges the performance gap between **frontier orchestration models** and **local deployable systems**.
It enables:

- **Privacy-preserving AI agents** that run fully offline.
- **Efficient orchestration** without reasoning latency overhead.
- **Scalable, modular training** for tool ecosystems that evolve over time.

By decoupling fine-tuning and introducing dynamic adapter loading, DualTune democratizes **agentic AI**, bringing practical autonomy to the edge.

## Code and Contributions:

## Contact:

Please contact us () to integrate DualTune into your agentic systems.