

LLM Booster SDK Developer Edition

Installation, command line, and Python API

Developer Edition license: This edition is for evaluation and development only. Production/commercial use and redistribution are not licensed. Features are not technically disabled. Review the included LICENSE-DEVELOPER.md before use.

Overview: LLM Booster prepares, routes, and cleans agent context locally and provides experimental transport-coding helpers. The default SDK and CLI flows do not call an LLM service.

1. Install from the Developer Edition ZIP

Python 3.10 or newer is required. Download the private Developer Edition ZIP from the distribution location and run these commands in the folder containing the archive. They extract the package, create an isolated environment, install the included wheel, and check the CLI:

```
Expand-Archive .\llm_booster_developer_edition-0.1.0.zip .\llm-booster-dev
Set-Location .\llm-booster-dev
py -0p
py -3.10 -m venv .venv
$wheel = Get-ChildItem -Filter 'llm_booster-*.whl' | Select-Object -First 1
if (-not $wheel) { throw "Developer Edition wheel not found." }
.\.venv\Scripts\python.exe -m pip install $wheel.FullName
$cli = ".\venv\Scripts\llm-booster.exe"
& $cli --help
```

The ZIP contains the wheel, license, README, example context, and Turkish and English guides. The example JSON file is at examples/context.json. For editable development from the source checkout instead, create a virtual environment and run `python -m pip install -e .` from the repository root.

2. Run from the command line

The CLI reads a UTF-8 JSON object that maps agent names to context text. The packaged sample file is examples/context.json. Commands below use the installed executable in the virtual environment:

```
$cli = ".\venv\Scripts\llm-booster.exe"
& $cli run --mode baseline --request "Check the invoice" --context examples/context.json
& $cli run --mode cleanup --request "Check the invoice" --context examples/context.json
& $cli run --mode semantic --top-k 2 --request "Check the invoice" --context examples/context.
json
& $cli run --mode linear_network_coding --request "Check the invoice" --context examples/
context.json
```

Modes

Mode	Behavior
baseline	All configured agents; preserves original context unchanged.
cleanup	Default. Trims line edges and removes blank and exact duplicate lines.
semantic	Selects the Top-K keyword-ranked agents and cleans their context.
linear_network_coding	Preserves text and adds GF(2) source/XOR parity packets; tolerates any one packet loss.

Output is indented JSON on stdout. Invalid JSON, an unknown agent key, or an invalid option produces an error and a nonzero exit code. Use `--agents agents.json` to pass a JSON list with unique agent names and optional descriptions and skills.

3. Python API

Booster exposes the same four modes to application code:

```
from llm_booster import Booster

booster = Booster()
context = {
    "billing_agent": "Invoice 426 TL is not paid.\n"
                    "Invoice 426 TL is not paid.",
    "device_agent": "SIM 123 is active.",
}
result = booster.run("Check the invoice", context, mode="cleanup")
print(result["selected_agents"])
print(result["context"])
print(result["metrics"])
```

For semantic routing, configure meaningful agent names, descriptions, and skills. Context keys must match the configured agent names. `top_k` must be a positive integer; keyword overlap is not an embedding or model-based semantic score.

4. Local models through Ollama

The CLI and `Booster.run` only prepare context; they do not invoke a model. To run local inference, install and start Ollama, pull a model, and explicitly call the SDK's Ollama helper. Model downloads require network access and disk space; inference is sent to the local Ollama API at `http://localhost:11434`.

```
ollama pull qwen2.5:3b
ollama pull nemotron-mini:4b

from llm_booster import call_local_ollama

result = call_local_ollama(
    "Answer in Turkish: What should I check on invoice 426 TL?",
    model="qwen2.5:3b", # Or "nemotron-mini:4b" / "kumru:2b".
)
if not result["ok"]:
    raise RuntimeError(result["error"])
print(result["text"])
```

For Kumru 2B, the institution guide explains how to download a GGUF file and create a local Ollama model. Model weights may be free to download, but hardware has a cost; review each model's license and usage terms separately. Nemotron Mini 4B is optimized for English, so assess its Turkish output before use.

5. Context coding and GF(256) experiment

`linear_network_coding` uses the GF(2) systematic XOR codec. It adds one parity packet and can recover any single erased packet; decoding requires the packet metadata.

```
from llm_booster import Booster, decode_context

result = Booster().run(
    "Check the invoice",
    {"billing_agent": "Invoice 426 TL is unpaid."},
    mode="linear_network_coding",
)
bundle = result["coded_context"]
bundle["packets"].pop(0) # Simulate a single packet loss.
recovered = decode_context(bundle)
```

`rlnc_roundtrip` is a separate GF(256) benchmark helper for byte payloads. It requires all coded blocks and is not a packet loss recovery API. Neither experiment claims token savings; coding adds transport overhead and provides no encryption or corruption detection.

6. Validation and troubleshooting

```
# Available in the source checkout, not in the wheel-only ZIP:
.\.venv\Scripts\python.exe -m unittest discover -s tests -v
```

- If the CLI is not found, use the virtual-environment executable shown above.

- For unknown-agent errors, ensure context keys match configured agent names exactly.
- For invalid JSON or unreadable files, verify the path and UTF-8 JSON object format.
- Insufficient independent packets means the available parity cannot recover the loss; preserve the complete packet set and metadata.

Package version: 0.1.0. This guide describes the Developer Edition wheel distributed in the accompanying ZIP.