DOCUMENTATION CELLULE-PRO

Mode Outils

Function-calling · tool-use OpenAI compat

cellule.ai · pro.cellule.ai
2026-05-18

Table des matières

0.1 Tools-mode — routing universel pour clients tool-calling

Cellule-PRO route automatiquement les requêtes tool-calling (`POST /v1/chat/completions` avec `tools=[...]`) vers les workers **tool-fluent** du parc, c'est-à-dire ceux qui savent enchaîner plusieurs `tool_calls` JSON OpenAI conformes au schéma.

Cette page documente le mécanisme, les modèles recommandés, la surveillance et l'override admin.

1. Pourquoi un mécanisme dédié

Tous les LLM ne savent **pas** faire du tool-calling fiable. Sur un prompt simple en 1 step (`get_weather('Paris')`), un Qwen 3.5 9B émet un `tool_call` valide. Mais sur un prompt en 2+ steps (`list_dir → write_file`, typique d'un agent coding type opencode/ Cursor/Continue), le 9B **déraille au step 2** : il préfère générer le code en markdown dans `content` plutôt que d'appeler le tool.

Doctrine validée David 2026-04-29 : il faut router les requêtes tools-mode vers les **modèles tool-fluent runtime-validés**, pas sur un seuil grossier `params_b  7B`.

2. Détection runtime — pas de hardcode par modèle

Cellule-PRO **ne hardcode pas** la liste des modèles tool-capable. L'image livrable est universelle : un client peut connecter un worker avec n'importe quel GGUF (Llama, DeepSeek, Mistral, Phi, Gemma, Qwen, etc.). Le pool **probe** chaque worker au handshake via un tool-bench V2 standardisé.

Bench V2 multi-step

Le pool envoie au worker au boot une conversation 2 tours :

1. **Step 1** : prompt `"Use list_dir to see what's in /tmp, then use write_file to save /tmp/bench.txt with content 'hello world'."`
   - 2 tools `list_dir` et `write_file`. Le worker doit émettre `tool_calls=[list_dir(/tmp)]`.
2. **Step 2** : le pool injecte un faux résultat de `list_dir` (`bench_existing.txt\nworkfile.json\nlogs/`) et redemande au worker de continuer. Le worker doit émettre `tool_calls=[write_file(/tmp/bench.txt, ...)]`.

Résultat	Verdict	
Step 1 OK + Step 2 OK	`tools_capable=TRUE`	
Step 1 OK + Step 2 markdown	`tools_capable=FALSE`	(typique 9B)
Step 1 sans tool_call	`tools_capable=FALSE`	(typique 4B)
Step 1 timeout/erreur	`tools_capable=FALSE`	

Coût : ~50-100 tokens prompt × 2 + ~50 tokens completion × 2 = ~300 tokens par worker, ~5-10 sec selon le worker. Lancé en arrière-plan post-handshake, **non bloquant** pour le boot.

Persistence DB

Migration `032_tools_capable.sql` ajoute :

Colonne	Type	Sens
`tools_capable`	`BOOLEAN`	NULL=pas testé, TRUE=bench V2 OK, FALSE=fail
`tools_bench_at`	`TIMESTAMP`	Dernier bench (re-bench manuel/auto)
`tools_bench_score`	`REAL`	0.0 ou 1.0 (V1 : binaire)

3. Filtre routing

Quand une requête arrive avec `tools=[...]` (ou conv déjà sticky-tagged tools-mode, cf. rc57) :

```
# Pseudocode pool.get_available_worker
if require_tools_capable:
    candidates = [w for w in workers if w.info["tools_capable"] is True]
    # Strict : FALSE et NULL exclus
```

Le `params_b ≥ 7B` reste comme **guard de sécurité** (un 4B même faussement tagué OK ne servira pas un payload lourd).

Fail-open en bootstrap

Si **aucun** worker n'a encore passé son bench (post-restart pool, reconnects en série), le filtre `tools_capable=True` est temporairement relaxé : le pool retombe sur `params_b` floor seul. Évite les 503 systématiques pendant la fenêtre de bench (~5-30 sec). Le filtre redevient strict dès qu'au moins 1 worker a PASS.

Toggle env `ROUTING_TOOLS_REQUIRE_CAPABLE=on|off` (default `on`).

4. Modèles recommandés

Liste valide à la doctrine 2026-04-29 (sera mise à jour avec les tests futurs). Le bench V2 PASS sur :

Modèle	tools_capable bench V2
Qwen3-Coder-30B-A3B-Instruct (MoE)	(recommandé)
Qwen 3.6 35B-A3B (MoE)	(recommandé)
Qwen 2.5 32B-Instruct	(à valider)
Qwen 2.5 14B-Instruct	(à valider)
Qwen 2.5 Coder 7B-Instruct	(à valider — Coder mais petit)
Qwen 3.5 9B	tool-chaining déraille step 2
Qwen 2.5 7B-Instruct	(à bencher)
Qwen 3.5 4B / 2B	trop petit

Recommandation deploy DSI : pour activer opencode/Cursor/ Continue de manière fiable, prévoir au moins **un worker 30B-A3B+** dans le parc. Le pool route automatiquement les tool-calls vers lui ; les autres workers (9B/4B) servent les requêtes chat normales sans dégradation.

5. Surveiller le statut

Via dashboard admin

Onglet “Workers” → colonne **Tools** :

Badge	Sens
vert	`tools_capable=TRUE` (passe bench V2)
rouge	`tools_capable=FALSE`
gris	NULL — bench pas encore lancé ou en cours

(UI à livrer prochainement — la valeur DB est exposée dès rc58.)

Via SQL

```
SELECT worker_id, model_path, tools_capable, tools_bench_at, tools_bench_score
FROM workers
WHERE is_online = TRUE
ORDER BY tools_capable DESC NULLS LAST, worker_id;
```

Via logs

```
cellule.tools_checker  INFO  tools_bench Cellule-Coder-test: PASS (ok, 5.2s)
cellule.tools_checker  INFO  tools_bench Cellule-9B-Alpha-test: FAIL (step2 derail markdown ..., 8.1
```

6. Re-bench manuel

À la main via SQL (réinit la valeur, le pool re-bench au prochain handshake) :

```
UPDATE workers SET tools_capable = NULL, tools_bench_at = NULL, tools_bench_score = NULL
WHERE worker_id = 'Cellule-Coder-test';
```

Puis kicker le worker : `docker restart cellule-worker-XXX` ou laisser le pool re-bench au prochain heartbeat.

(Bouton “Re-bench” UI à livrer prochainement.)

7. Override admin

Si l’admin sait qu’un worker est tool-fluent mais que le bench V2 échoue (faux négatif) ou inversement :

```
-- Force TRUE (bypass bench)
UPDATE workers SET tools_capable = TRUE WHERE worker_id = 'My-Worker';

-- Force FALSE (exclu du tools-mode)
UPDATE workers SET tools_capable = FALSE WHERE worker_id = 'Buggy-Worker';
```

L'override est **persistant** : tant que le worker n'est pas re-bench manuellement (cf. §6), la valeur reste.

8. Configuration client opencode validée

```
{
  "$schema": "https://opencode.ai/config.json",
  "model": "cellule/CELLULE",
  "provider": {
    "cellule": {
      "npm": "@ai-sdk/openai-compatible",
      "name": "Cellule-PRO Pool",
      "options": {
        "baseURL": "http://<POOL_VIP>:18095/v1",
        "apiKey": "sk-cellule-XXXXXXXXXXXXXXXXXXXXXXXX"
      },
      "models": {
        "CELLULE": {
          "name": "Cellule-PRO Smart Pool",
          "limit": { "context": 131072, "output": 4096 }
        }
      }
    }
  }
}
```

Notes :

- `<POOL_VIP>` = adresse IP virtuelle du pool virtuel HA (chantier rc62), saisie au wizard first-boot dans l'étape *Réseau → Pool virtuel HA*. Tous les pools du cluster partagent cette VIP — si l'un tombe, un autre prend le relais en moins de 3 secondes via VRRP. Cf. ADMIN_GUIDE § "Pool virtuel HA". Si vous n'avez pas activé la VIP au wizard (déploiement mono-pool sans HA), utilisez l'IP du pool concret à la place.
- **Pas de** `"tools": true` dans le bloc model (le provider `@ai-sdk/openai-compatible` détecte automatiquement les tools natifs OpenAI). L'ajouter peut perturber la détection.
- `limit.context` doit refléter le ctx max de votre worker tool-fluent le plus large (ex. `131072` si vous avez un Coder 30B-A3B en `n_ctx=131072`).
- Le pool route automatiquement vers le worker tool-fluent — pas besoin de spécifier un model_id particulier côté client.

9. Limites connues

- **Bench V2 = harness 2 tours** : ne teste pas tools nested (objects imbriqués), arrays, énums, ni les boucles 5+ tours. Un modèle peut PASS V2 et fail sur un cas opencode très complexe. Pour ces cas, l'override admin (§7) à FALSE évacue le faux positif.

- **Re-bench n'est pas automatique** au model-change actuellement. Si un worker change de modèle (SELF-HEAL, update_model), le `tools_capable` ancien reste en RAM jusqu'au prochain restart du pool ou re-bench manuel. (TODO : invalidation auto.)
- **Aucun bench cross-langue** : le bench V2 prompt est en anglais. Un modèle qui PASS en anglais peut échouer en français. (TODO : prompt bilingue.)

10. Doctrine résumée

1. Cellule-PRO **détecte** tool-capability au runtime, **pas** par hardcode model-name.
2. Le routing tools-mode est **strict** : seul un worker prouvé PASS bench V2 reçoit du trafic tools.
3. Le DSI client n'a **rien à configurer** côté pool — il déploie un parc, le pool découvre quels workers sont tool-fluent.
4. Pour activer un agent coding fiable (opencode/Cursor/Continue) il **faut** un worker 30B-A3B+ dans le parc.
5. Si aucun worker tool-fluent disponible, le pool **fail-open** (mode rc59) : il sert quand même mais avec dégradation éventuelle ; observable via `tools_capable` colonne DB.