ChatGPT

Modèles de langage open source (< 2 milliards de paramètres) pour RAG local

Plusieurs modèles de langage de petite taille répondent aux critères d'un système *retrieval-augmented generation* (RAG) local : **multilingues (avec un bon français)**, **optimisés pour résumé et Q/R**, **légers pour CPU/GPU peu puissant**, et offrant un **équilibre qualité/vitesse**. Voici une comparaison des principaux candidats :

Comparaison des modèles

Modèle	Taille	Support linguistique	Optimisation tâches	Licence	Déploiement local
Qwen2-1.5B (Alibaba)	1,5 Md	Multilingue (fr, en, zh, +27) [1]	Instruct (chat, Q/R, code) – très haute qualité [2] [3]	Apache 2.0 [4]	CPU OK (32K contexte, GQA rapide) ; ~4 Go VRAM en FP16 (int8 dispo)
Salamandra-2B (BSC)	2,3 Md	Multilingue euro (35 langues incl. fr) [5]	Instruct (fine-tuné pour chat/Q/R) [6] – perf. compétitives similaires	Apache 2.0 [7]	CPU OK (quantifié recommandé : ~1,8 Go en 4-bit) [8] ; ~6 Go VRAM FP16
BLOOMZ-1.7B (BigScience)	1,7 Md	Multilingue (46 langues dont fr) [9]	Instruct multi-tâches (suivi d'instructions zero-shot) [10]	RAIL (usage responsable)	CPU OK (quant. int8 conseillé) ; ~4 Go VRAM FP16 (context 2048)
mT0-large (1.2B)	1,2 Md	Multilingue (plus de 40 langues) [11]	Instruct multitâche (dérivé de mT5) – bon en résumé & Q/R [12]	RAIL (BigScience)	CPU OK ; ~3 Go VRAM FP16 (archit. encodeur-décodeur)
FLAN-T5 Large (Google)	0,78 Md	Surtout anglais (comprendre fr limité)	Instruct (finetune 100+ tâches) – excellent résumé/Q-R en anglais [13]	Apache 2.0 [13]	CPU facile (modèle compact) ; ~2 Go VRAM FP16 (contexte ~512)

Tableau : *Principaux modèles open source <2B param., adaptés RAG.* Chaque modèle est **multilingue** (au moins anglais/français) et **optimisé** pour génération de réponses **concises et factuelles**. Tous sont utilisables sur **CPU ou petits GPU** (4–8 Go VRAM), surtout en versions *quantifiées* (int8, int4) pour réduire la latence et la mémoire.

Performance et qualité des réponses

Malgré leur petite taille, ces modèles offrent une **qualité de réponse surprenante** pour les tâches de résumé et de QA factuelle. Des benchmarks récents montrent que les **petits modèles** bien entraînés **peuvent atteindre une précision élevée** sur des Q/R basées document, proche de modèles 7B plus gros [2]. Par exemple, **Qwen2-1.5B** (1,5 Md) d'Alibaba, multilingue, affiche un niveau de performance **comparable à des modèles 7B de 2023** sur le question-réponse [2]. Il intègre des optimisations comme le **Grouped Query Attention (GQA)** pour accélérer l'inférence et gère un long contexte (jusqu'à 32k tokens) [14]. Qwen2-1.5B est considéré comme *« un excellent compromis taille vs qualité »*, offrant **rapidité sur CPU** et **réponses précises**, au point de rivaliser avec certains 7B [2] [15].

Le modèle **Salamandra-2B** du BSC, entraîné sur **35 langues européennes** (dont le français) [5], a également une version *instruct* orientée chat/Q-R. Les évaluations montrent qu'il obtient des résultats **proches des autres modèles ~2B** sur divers benchmarks multilingues [16]. Salamandra-2B excelle en compréhension du français et d'autres langues latines (entraînement dédié), et supporte un contexte de ~8k tokens. Son poids reste modéré (~2,3 Md paramètres) et il a été **quantifié efficacement** par ses auteurs (formats 4-bit Q4, Q5, etc.) pour minimiser l'impact sur la perplexité tout en réduisant la taille à ~1.7–1.9 Go [8]. En pratique, Salamandra quantifié tourne **aisément sur CPU** ou petits GPUs, avec une perte de qualité négligeable [17] [18].

Du côté de BigScience, **BLOOMZ-1.7B** et **mT0-1.2B** sont deux modèles ouverts orientés *instruction-following*. BLOOMZ est une version de BLOOM (modèle multilingue 46 langues) affiné sur des tâches multiturn en **plusieurs langues** [10]. Il est capable de suivre des instructions en français (par ex. *« Fais le résumé de ce texte… »*) ou en anglais sans fine-tuning supplémentaire, même si sa qualité en français est légèrement inférieure à l'anglais d'après les auteurs [19]. mT0-large, lui, est dérivé du modèle T5 multilingue (mT5) et fine-tuné sur le même corpus d'instructions multilingues (*xP3*). Tous deux affichent une **excellente performance zero-shot** sur des tâches comme XNLI, QA multilingue et résumé, surpassant largement les anciens T5/BART non-instruits [19]. Ils tendent toutefois à parfois répondre en anglais par défaut si non spécifié (BLOOMZ) [19]. Leur **couverture linguistique** est très étendue (dizaines de langues), avec une bonne maîtrise du français (corpus OSCAR et divers data multilingues utilisés).

Le modèle **FLAN-T5 Large** (780 M) se démarque par sa **vitesse** et son efficacité en résumé/Q-R, surtout en anglais. Issu de la fine-tune FLAN de Google, il a appris sur plus de 100 tâches d'instructions diverses, ce qui le rend polyvalent et capable de **suivre des consignes complexes** [13]. En français, il n'est pas explicitement entraîné, mais on observe qu'il peut souvent comprendre et répondre de manière basique si la consigne est simple (au besoin, on peut le *fine-tuner* sur des données françaises spécifiques pour améliorer). FLAN-T5 est un modèle encodeur-décodeur, idéal pour des tâches de génération structurée comme le résumé. Sa taille modeste lui confère une latence très faible et une exigence mémoire minime – il peut tourner **en quelques secondes sur CPU** pour résumer un paragraphe court, avec une qualité approchant BART-large sur CNN/DailyMail en anglais.

En résumé, ces modèles équilibrent **qualité** et **vitesse** : bien que de plus petite taille, ils fournissent des réponses généralement cohérentes et précises sur du *QA extractif* ou du résumé factuel, à condition de **fournir le contexte pertinent** via le retrieveur. Les dernières itérations (ex : Qwen2, phi-1.5) montrent

que l'**écart avec les modèles 7B** s'est réduit – par exemple, *« Qwen-2 1.5B offre une combinaison taille/qualité étonnante, suffisant pour la plupart des tâches factuelles avec un pipeline bien conçu »* [2] [15]. Naturellement, ils peuvent encore halluciner ou faire des erreurs subtiles, mais dans un cadre RAG (où le texte source est fourni), ils atteignent souvent **>95 % de précision** sur des questions standards [20] [2], notamment si on les a spécialisés via fine-tuning supplémentaire.

Intégrations et projets récents en RAG

Plusieurs projets open-source démontrent l'usage de ces petits modèles dans des systèmes RAG complets :

- **Haystack (deepset)** : Ce framework QA intègre un composant de génération local via *HuggingFace Transformers*. On peut par exemple configurer Haystack pour utiliser un modèle comme *FLAN-T5-large* en génération de réponse après retrieval [21]. Des tutoriels montrent la construction d'un *chatbot* RAG privé combinant Haystack, Prompt-box et FLAN-T5 pour résumer des documents internes [22]. Haystack supporte également d'autres modèles HuggingFace – BLOOMZ, mT0 ou DistilBART – pour le résumé multi-documents.

- **Danswer** : Plateforme Q/R open-source, Danswer permet d'utiliser des LLM locaux (*via Ollama, GPT4All, etc.*) pour générer les réponses à partir des documents indexés. Par défaut, Danswer utilise GPT-3.5 via API, mais on peut le pointer sur un modèle comme **GPT4All** quantifié. Par exemple, la doc Danswer montre l'utilisation d'un modèle GPT4All basé sur *Mistral-7B* quantifié (openorca 4-bit) pour un déploiement 100% local [23] [24]. Il ajuste alors certains paramètres (timeout, prompts) pour compenser la lenteur relative du modèle CPU. Cette intégration prouve qu'un modèle ~7B compressé peut fonctionner en RAG local, et on peut de même employer un **Salamandra-2B Q4** ou **Qwen-1.5B int8** de façon analogue pour un meilleur temps de réponse.

- **LLMware (AI Bloks)** : Ce projet de recherche a lancé la série de modèles **BLING/DRAGON**, justement destinée aux petits LLM optimisés pour RAG [25]. Ils ont fine-tuné des modèles 0.5B–3B (bases Pythia, Falcon, Cerebras, etc.) sur des données de QA factuelle, extraction de données, résumés bullet points, etc., afin de mesurer jusqu'où on peut aller avec un *small LLM* dans un pipeline RAG [26]. Leurs benchmarks montrent qu'un modèle de ~1,3 Md bien entraîné peut fournir des réponses correctes dans ~98 % des cas sur un test Q/R fermé, contre ~100 % pour les meilleurs modèles 7–13B [27] [2]. Cela confirme le potentiel des *petits modèles instruits* pour des applications d'entreprise, surtout quand ils sont intégrés dans un pipeline robuste (retriever performant, post-traitement, etc.) [28].

- **Autres** : Citons également *PrivateGPT* (script populaire intégrant LlamaCpp + GPT-J 6B quantifié pour interroger des fichiers locaux) ou *PaperQA* [29], qui montrent l'intérêt de LLM locaux pour interroger des données privées. De son côté, la communauté *Ollama* fournit des conteneurs prêts à l'emploi pour déployer, sur CPU, des modèles comme **Salamandra-2B-Instruct quantifié** ou des variantes distillées de Llama2, facilitant la mise en place rapide d'un chatbot RAG sur sa machine personnelle [30] [31].

En somme, l'écosystème RAG s'adapte de plus en plus à ces modèles légers : ils sont suffisamment **rapides** (quelques tokens par seconde sur CPU) et **peu coûteux en mémoire** pour être utilisés dans des serveurs locaux ou même embarqués, tout en maintenant une qualité acceptable pour de nombreux cas d'usage (*assistant documentaire, recherche d'info, etc.*).

Quantification et distillation pour utilisation sur CPU

Pour maximiser la vitesse d'inférence sur CPU, il est souvent judicieux de recourir à la **quantification** ou à la **distillation** des modèles :

- **Quantification** : Réduire la précision des poids (de 16/32 bits vers 8, 4, voire 2 bits) diminue drastiquement la mémoire nécessaire et accélère les calculs. La plupart des modèles cités supportent une quantification 8-bit sans perte notable de qualité. Par exemple, Salamandra-2B a été quantifié en plusieurs formats par ses auteurs : en **4-bit (Q4_K)** le modèle instruct ne pèse plus que ~1,8 Go avec seulement une légère dégradation perplexité [8]. De même, Qwen-1.5B quantifié en int4 tiendra en ~1,2 Go de RAM. Des outils comme `llama.cpp`, `GPTQ` ou Transformers `bitsandbytes` permettent d'obtenir facilement ces versions compressées, compatibles CPU. D'après les tests, les quantifications 4-bit les plus avancées (ex : Q4_K_M) sont **quasi-équivalentes** à l'original (perte <1-2 % de score) [32]. **Conclusion** : en quantifiant, on peut faire tourner un modèle ~2B sur un laptop CPU 8 Go avec une latence tout à fait raisonnable (quelques secondes pour une réponse courte).

- **Distillation** : Il s'agit d'entraîner un **modèle plus petit** à imiter un grand modèle, pour obtenir un *student model* plus compact mais presque aussi compétent. Google a ainsi distillé T5-XXL en *Tiny T5* pour créer FLAN-T5, et des recherches de Microsoft ont donné **Phi-1.5** (1,3 Md) qui **égale ou surpasse Llama2-7B** sur certains benchmarks math/code [33] [34]. Concrètement, on peut par exemple **distiller un T5 Large vers un T5 Small** sur la tâche de résumé CNN/DailyMail, en perdant <2 points ROUGE [35] [36]. HuggingFace propose déjà des versions distillées de BART pour le **summarization** (e.g. *DistilBART* 140M entraîné sur CNN/DM) qui conservent ~97% de la qualité de BART-large tout en doublant la vitesse [37] [38]. Dans le contexte RAG, la distillation peut aussi se faire *en condition supervisée* : on génère des réponses avec un modèle puissant (ex : Llama-2-13B) sur le corpus cible, puis on fine-tune le petit modèle sur ces paires contexte→réponse. Cette approche permet d'améliorer la pertinence du petit modèle sur le domaine en question.

En pratique, **combiner quantification et distillation** donne les meilleurs résultats pour le déploiement CPU. Par exemple, un FLAN-T5-large distillé en « mini-FLAN » 250M et quantifié 8-bit pourrait fournir des réponses en une fraction de seconde sur CPU, avec une qualité suffisante pour de la FAQ basique. De même, les travaux BLING suggèrent qu'un modèle 1,3B spécialisé peut atteindre ~98 % de précision sur du QA fermé [2] – ce qui, pour beaucoup d'applications professionnelles, est *assez bon* compte tenu des **coûts et latences réduits** (modèles 10× à 100× moins lourds à héberger) [39].

En conclusion, il existe aujourd'hui plusieurs *petits modèles open source* répondant aux besoins RAG en français : ils sont **multilingues**, **performants en résumé et Q/R**, et **facilement déployables en local** sur du matériel modeste. Des initiatives comme Qwen-2 1.5B ou Salamandra 2B montrent que la barrière des 7B n'est plus absolue : on peut obtenir des réponses de haute qualité avec <2B paramètres, surtout en exploitant la synergie avec un bon module de retrieval. Pour tirer le meilleur parti de ces modèles, on préconise d'utiliser la **quantification** (int8/int4) et éventuellement la **distillation**, afin d'atteindre la **faible latence** souhaitée tout en maintenant un excellent **niveau de réponse** pour vos cas d'usage en local.

Sources :

- BSC, *Salamandra Technical Report* – modèle 2B multilingue (35 langues) + quantification [5] [8]
- Qwen Team (Alibaba), *Hello Qwen2* – annonce de Qwen2-1.5B (27 langues, Apache 2.0) [3] [4]

- D. Oberst, *Benchmarking Small LLMs (BLING)* – résultats RAG : petits modèles vs 7B [2] [15]
- BigScience, *BLOOMZ/mT0: Multitask Finetuning for Multilingual LLMs* [19] [12]
- Haystack Documentation – usage de Flan-T5 large pour QA local [21]
- TechLatest, *Danswer Configuration* – support de GPT4All/Mistral 7B en local [23] [24]
- B. Burtenshaw, *Tiny LMs: Summarising at low latency* – distillation T5 pour résumé [35] [38]

---

[1] [3] [4] [14] Hello Qwen2 | Qwen
https://qwenlm.github.io/blog/qwen2/

[2] [15] [20] [25] [27] [28] [39] Best Small Language Models for Accuracy and Enterprise Use Cases — Benchmark Results | by Darren Oberst | Medium
https://medium.com/@darrenoberst/best-small-language-models-for-accuracy-and-enterprise-use-cases-benchmark-results-cf71964759c8

[5] [16] Salamandra Technical Report
https://arxiv.org/html/2502.08489v2

[6] [7] [8] [17] [18] [30] [31] [32] robbiemu/salamandra:2b-instruct_Q3_K_L
https://ollama.com/robbiemu/salamandra:2b-instruct_Q3_K_L

[9] You Can No Longer Fail To Understand How To Use Large ...
https://towardsai.net/p/l/you-can-no-longer-fail-to-understand-how-to-use-large-language-models

[10] bigscience/bloomz-1b7 - Hugging Face
https://huggingface.co/bigscience/bloomz-1b7

[11] [12] Bloomz Mt · Models - Dataloop
https://dataloop.ai/library/model/bigscience_bloomz-mt/

[13] Running the Large Language Model FLAN-T5 locally | Niklas Heidloff
https://heidloff.net/article/running-llm-flan-t5-locally/

[19] Papers Explained 99: BLOOMZ, mT0 - Ritvik Rastogi
https://ritvik19.medium.com/papers-explained-99-bloomz-mt0-8932577dcd1d

[21] Chunking For LLMs Using Haystack & HuggingFace - Cobus Greyling
https://cobusgreyling.medium.com/chunking-for-llms-using-haystack-huggingface-81255776d71

[22] Improving RAG by Optimizing Retrieval and Reranking Models
https://docs.v1.argilla.io/en/v1.24.0/tutorials_and_integrations/tutorials/feedback/fine-tuning-sentencesimilarity-rag.html

[23] [24] Exploring AI Configuration & Settings in Danswer | Medium
https://medium.com/@techlatest.net/exploring-ai-configuration-settings-in-danswer-gpt-to-chat-query-your-own-data-part-1-dc3d3b303505

[26] Small Instruct-Following LLMs for RAG Use Case - Payble - AI SAAS
https://llmware.ai/resources/small-instruct-following-llms-for-rag-use-case

[29] Future-House/paper-qa: High accuracy RAG for answering ... - GitHub
https://github.com/Future-House/paper-qa

[33] Microsoft phi-1.5: a 1.3B model with performance comparable to ...
https://www.reddit.com/r/mlscaling/comments/16geytf/microsoft_phi15_a_13b_model_with_performance/

[34] Textbooks Are All You Need: Microsoft PHI-1.5 Model with 1.3 Billion ...
https://youssefh.substack.com/p/textbooks-are-all-you-need-microsoft?utm_source=profile&utm_medium=reader2

[35] [36] [37] [38] Tiny Language Models: Summarising text at low latencies | by Ben Burtenshaw | Medium
https://medium.com/@ben.burtenshaw/tiny-language-models-summarising-text-at-low-latencies-c1fbab509673