Modelos de Linguagem de Grande Porte (LLMs) e Otimização de Contexto

Artigos Fundamentais sobre LLMs

Para compreender o funcionamento e a estrutura dos **modelos de linguagem de grande porte (LLMs)**, é importante revisar trabalhos fundamentais que marcaram a evolução dessas tecnologias. Um ponto de partida é o artigo **"Attention Is All You Need" (Vaswani et al., 2017)**, que introduziu a arquitetura *Transformer*. Esse trabalho mostrou que é possível substituir redes recorrentes por mecanismos de **auto-atenção multi-cabeças**, eliminando recursões e convoluções, e ainda assim obter melhor desempenho e *throughput* de treinamento. Essa arquitetura Transformer tornou-se a base da maioria dos LLMs modernos, habilitando o processamento paralelo de sequências e a captura de dependências de longo alcance no texto.

Outro marco foi a emergência dos **modelos pré-treinados em larga escala**. Dois paradigmas principais surgiram: modelos do tipo *encoder-decoder* (p.ex. **BART**, **T5**) e modelos *decoder-only* autoregressivos (p.ex. **GPT**). O trabalho **GPT-3 (Brown et al., 2020)** demonstrou que ao escalar um modelo autoregressivo para centenas de bilhões de parâmetros e treiná-lo em um volume massivo de texto, o modelo adquire a capacidade de **aprendizado por poucos exemplos (*few-shot learning*)**, conseguindo generalizar para novas tarefas apenas com instruções ou exemplos no prompt. Esse fenômeno evidenciou as *capacidades emergentes* dos LLMs conforme se aumenta o tamanho e os dados de treinamento. Já os modelos do tipo *encoder-only*, como o **BERT (Devlin et al., 2018)**, introduziram o conceito de pré-treinamento por **modelagem de linguagem mascarada**, focando na compreensão de linguagem e obtendo avanços em tarefas de classificação e resposta em contexto limitado.

Mais recentemente, a disponibilidade de modelos *open-source* treinados com alta qualidade tornou viável a pesquisa reprodutível em LLMs. O artigo da **Meta AI sobre o LLaMA (Touvron et al., 2023)** é um exemplo disso: apresentou uma família de modelos de 7 a 65 bilhões de parâmetros, treinados somente com dados públicos, que atingem desempenho de ponta. O **LLaMA-13B** conseguiu superar o GPT-3 (175B) em muitos *benchmarks*, enquanto o **LLaMA-65B** foi competitivo em nível de modelos maiores como o Chinchilla-70B da DeepMind. A disponibilização desses modelos para a comunidade permitiu experimentação aberta, inclusive em ambientes otimizados como o *Llama.cpp*.

Em resumo, para entender LLMs é fundamental conhecer: (1) a **arquitetura Transformer** e mecanismos de atenção, (2) os **paradigmas de pré-treinamento** (mascarado vs. autoregressivo) exemplificados por BERT e GPT, (3) o impacto do **escalonamento de tamanho e dados** evidenciado por GPT-3, e (4) os **modelos foundation open-source** como LLaMA que viabilizam pesquisa prática. Esses trabalhos formam a base conceitual sobre a qual se desenvolvem técnicas mais especializadas, como as de otimização de contexto discutidas na próxima seção.

Técnicas de Otimização de Contexto em LLMs

*Exemplo de pipeline de **Recuperação de Contexto** com **Re-Ranking** e **Compressão Contextual***. *Um modelo de busca primeiro recupera um conjunto inicial de fragmentos relevantes (chunks) de documentos para uma consulta do usuário. Em seguida, um módulo de re-rank (reordenamento) com possíveis técnicas de compressão de contexto refina esses fragmentos, produzindo um subconjunto menor e mais pertinente de informações que será fornecido ao LLM para gerar a resposta final.*

LLMs possuem **janelas de contexto limitadas** (isto é, podem processar apenas um número máximo de tokens de entrada). Por isso, otimizar o uso do contexto é crucial quando trabalhamos com documentos longos ou bases de conhecimento extensas. A seguir, são apresentadas as principais técnicas de otimização de contexto, com explicações e comparativos:

Recuperação de Contexto (*Context Retrieval*)

A recuperação de contexto envolve buscar informações externas relevantes à consulta do usuário para suplementar o prompt do LLM. Esse método é a base do paradigma **RAG (*Retrieval-Augmented Generation*)**, em que o modelo primeiro **recupera documentos ou passagens relevantes** de uma base de conhecimento e depois os utiliza para gerar uma resposta. Essa abordagem permite que LLMs acessem **conhecimento de longo cauda e informações atualizadas** sem precisar incorporá-las nos pesos do modelo. Por exemplo, um modelo como LLaMA pode buscar artigos da Wikipedia relacionados a uma pergunta e, então, condicionar sua geração nessas passagens recuperadas. As vantagens da recuperação incluem manter o modelo compacto (delegando o armazenamento de conhecimento a uma base externa) e a possibilidade de **atualizar informações dinamicamente**. No entanto, a eficácia depende de um bom índice e método de busca; recuperações imprecisas podem trazer *ruído* (informação irrelevante) para o modelo, reduzindo a qualidade da resposta. Assim, muitas vezes a recuperação é combinada com as técnicas a seguir para garantir que apenas conteúdos úteis sejam efetivamente usados.

Fragmentação de Documentos (*Document Chunking*)

"Chunking" é a técnica de **quebrar documentos longos em segmentos menores** (*chunks*) para viabilizar sua indexação, recuperação e entrada no modelo dentro dos limites de contexto. Cada segmento deve ser suficientemente autocontido: *"como regra geral, se um trecho de texto faz sentido por si só (sem muito contexto adjacente) para um humano, então fará sentido para o modelo de linguagem"*. A fragmentação adequada aumenta a **relevância dos resultados recuperados**, pois evita que trechos não relacionados (ou muito extensos) sejam considerados conjuntamente. Por exemplo, ao indexar um manual extenso, pode-se dividi-lo por seções ou parágrafos para que a busca retorne apenas partes diretamente ligadas à consulta. Uma vantagem clara do chunking é contornar a **limitação da janela de tokens**: ao dividir o conhecimento em pedaços menores, podemos recuperar apenas aqueles mais relevantes e encaixá-los no prompt. Em contrapartida, se os *chunks* forem muito pequenos, corre-se o risco de perder contexto necessário à compreensão, e se forem muito grandes, podem **diluir a relevância** com conteúdo irrelevante. Portanto, existe um equilíbrio a ser atingido no tamanho dos segmentos. Estudos práticos sugerem experimentar tamanhos típicos entre ~100 a 500 tokens por fragmento, dependendo do modelo e tarefa.

Re-Ranking de Passagens (*Passage Re-ranking*)

Após a recuperação inicial de documentos ou passagens, é comum aplicar um passo de **re-rank**, ou reordenação, para **priorizar os conteúdos mais relevantes** para a consulta. O re-ranking geralmente utiliza modelos de linguagem ou modelos especializados para analisar cada *chunk* recuperado em relação à pergunta do usuário e atribuir um **escore de relevância semântico**. Diferentemente da busca inicial (que pode ser uma busca lexical como BM25 ou vetorial por similaridade de embeddings), o re-ranker aprofunda a análise contextual, identificando quais trechos realmente respondem à pergunta. Na prática, isso pode ser implementado com um transformador *cross-encoder* que recebe a pergunta e o texto do candidato e produz uma pontuação de relevância. Essa etapa **melhora significativamente a qualidade** do contexto fornecido ao LLM, indo além da correspondência por palavras-chave para entender significado e contexto. A desvantagem é o custo computacional extra: o re-ranker é um modelo adicional avaliado sobre múltiplos candidatos, o que pode ser pesado se muitos documentos forem considerados. Por isso, costuma-se fazer um **retrieval inicial rápido** (retornando *N* candidatos) seguido de re-rank em um subconjunto menor (por exemplo, reordenar os top 50 para selecionar os top 5 finais). Implementações eficientes incluem re-rankers leves ou *fine-tuning* de um LLM menor apenas nas primeiras camadas. Em suma, o re-ranking **assegura que o LLM trabalhe com informação de alta qualidade**, reduzindo o ruído e potencializando a exatidão da geração final.

Chamada de Ferramentas Externas (*Tool Calling*)

Embora LLMs apresentem habilidades notáveis, eles ainda têm limitações em certos domínios – por exemplo, realizar **cálculos matemáticos exatos**, acessar **informações atualizadas** ou consultar **serviços externos** (tradução, calendários, bancos de dados). A técnica de *tool calling* procura mitigar essas deficiências permitindo que o modelo **utilize ferramentas ou APIs externas** durante o processo de geração. Um exemplo seminal é o **Toolformer (Schick et al., 2023)**, onde o próprio modelo aprende, de forma auto-supervisionada, *quando e como* invocar APIs externas (como calculadora, buscador web, tradutor) e incorporar os resultados em seu texto. Com apenas algumas demonstrações de uso de cada ferramenta, o modelo desenvolve a capacidade de decidir quais ferramentas são úteis para uma determinada tarefa e integrar suas saídas na resposta. O resultado é que um LLM relativamente menor pode **alcançar desempenho próximo a modelos bem maiores** em tarefas que envolvem essas funções especializadas. A vantagem do tool calling é clara: o LLM **"delega" subtarefas** – por exemplo, ao invés de tentar resolver um cálculo complexo (que poderia levar a erros de arredondamento ou alucinações), ele chama uma calculadora; para buscar um fato atualizado, ele consulta um mecanismo de busca, etc. Assim, combina-se a compreensão de linguagem do LLM com a **precisão ou frescor de ferramentas dedicadas**. Em termos de implementação, um desafio é treinar ou instruir o modelo a fazer essas chamadas de maneira coerente (por meio de *prompts* especiais ou *fine-tuning*). Além disso, requer uma infraestrutura para executar as ferramentas e fornecer os resultados ao modelo em tempo real. Do ponto de vista da compatibilidade, ferramentas podem ser integradas em pipelines com modelos open-source (inclusive rodando localmente via `llama.cpp`), desde que haja um **orquestrador** capaz de interceptar a consulta do modelo e produzir a resposta da ferramenta inserida no contexto.

Compressão e Truncamento de Contexto

Por fim, quando é inevitável lidar com contextos muito grandes (textos longos ou muitos documentos relevantes), entram em jogo técnicas de **compressão** ou **resumo de contexto**. O objetivo aqui é **representar a informação essencial de um contexto extenso de forma mais compacta**, cabendo na janela do modelo. Abordagens simples incluem gerar **sumários** dos documentos ou filtrar partes menos relevantes (truncamento seletivo). Abordagens mais avançadas envolvem representações latentes: por exemplo, condensa-se o contexto em um conjunto de *embeddings* ou em um *soft prompt* treinado que será fornecido ao modelo no lugar do texto original. Trabalhos recentes exploram essa ideia de diferentes modos. O método **xRAG (Cheng et al., 2024)** propõe aproveitar os *embeddings* já gerados por um modelo de recuperação denso e inseri-los diretamente no espaço de representação do LLM, efetivamente eliminando a necessidade de inserir todo o texto correspondente. Em xRAG, apenas uma ponte de modalidade treinável é ajustada; o LLM e o sistema de recuperação permanecem inalterados, mantendo um esquema *plug-and-play*. Os resultados mostram que essa integração pode **reduzir drasticamente o volume de texto necessário** (taxa de compressão extrema, chegando a representar documentos inteiros por poucos tokens embeddings) com **quase nenhuma perda de desempenho**. Outra linha de pesquisa, exemplificada pelo **In-Context Former (Wang et al., 2024)**, utiliza um modelo auxiliar para **gerar um "prompt digerido"** – um conjunto de vetores aprensíveis que condensa um documento longo – o qual substitui o texto original na entrada do LLM. Essas técnicas de compressão de contexto partem do princípio de que **textos longos contêm muitas redundâncias**, então é possível extrair deles uma essência informativa menor sem degradar a resposta. A vantagem evidente é possibilitar à LLM considerar um volume maior de conhecimento do que suportaria originalmente, **mantendo tempo de inferência e custo computacional baixos**. Por exemplo, xRAG relatou alcançar uma redução de 3,5 vezes nos FLOPs totais, mantendo desempenho similar ao caso sem compressão em vários datasets. A desvantagem é que a compressão pode falhar em preservar algum detalhe importante se não for bem feita – há sempre um *trade-off* entre compressão e fidelidade. Além disso, a implementação costuma exigir um estágio de treinamento ou ajuste fino do compressor (seja uma rede de sumarização ou um módulo gerador de prompt). Em cenários menos complexos, abordagens heurísticas como resumos por parágrafo ou truncar partes menos relevantes (por exemplo, últimos parágrafos em textos ordenados por relevância) também podem ser empregadas como soluções rápidas.

Comparativo das Técnicas: Cada técnica abordada traz benefícios específicos e desafios de implementação. A tabela a seguir resume as principais características de cada técnica de otimização de contexto discutida, incluindo vantagens, desvantagens, complexidade e considerações de uso com modelos compatíveis com **Llama.cpp** (ou seja, LLMs open-source executados localmente):

Técnica	Vantagens	Desvantagens	Complexidade	Compatibilidade (Llama.cpp)
Recuperação de Contexto (*Context Retrieval*)	– Permite acesso a conhecimento externo atualizado e especializado. – Mantém o modelo enxuto (conhecimento fica fora dos pesos).	– Requer uma base de dados e índice bem construído. – Pode introduzir "ruído" se os resultados não forem relevantes.	Pipelines de busca vetorial ou lexical; necessita ajuste fino de embeddings e infraestrutura de banco de dados.	**Sim.** Pode ser implementada externamente ao modelo (ex.: via biblioteca de busca), alimentando textos ao LLM open-source.
Fragmentação (Chunking)	– Garante que conteúdos caibam na janela de contexto do LLM. – Aumenta a precisão da busca ao focar em partes autocontidas.	– Escolha subótima do tamanho de chunk pode causar perda de contexto ou inclusão de irrelevâncias. – Exige preprocessamento de todo o corpus.	Relativamente simples (pré-processamento offline); envolve experimentação para definir tamanho e estratégia de separação.	**Sim.** Totalmente externa ao modelo – prepara os dados para uso eficiente com qualquer LLM.
Re-Ranking de Passagens	– Melhora a relevância do contexto usado pelo LLM, filtrando informação distrativa. – Aumenta a precisão e qualidade das respostas geradas.	– Adiciona custo computacional (modelo extra para pontuar passagens). – Precisa de dados rotulados ou heurística para treinar ou configurar o rankeador.	Moderada: pode usar modelos já existentes (p.ex., cross-encoder BERT) ou *fine-tuning* leve de LLMs menores.	**Sim.** Pode-se usar modelos open-source (ex.: MiniLM, BERT) para re-rank, integrando com LLM principal via código.
Chamada de Ferramentas (*Tool Calling*)	– Estende as capacidades do LLM (cálculo preciso, info em tempo real, etc.). – Melhora desempenho em tarefas que LLM sozinho teria dificuldade, sem precisar modelo maior.	– Implementação complexa: requer orquestração das chamadas e possivelmente *fine-tuning* do LLM para inserir e usar resultados. – Depende de ferramentas externas (ex.: acesso à internet, APIs).	Alta: envolve desenvolvimento de um agente ou modificação no decodificador do LLM para incorporar chamadas (ex.: frameworks estilo LangChain).	**Sim.** Ferramentas podem ser invocadas por agentes externos controlando um LLM local (ex.: usar `llama.cpp` + código Python para calculadora, busca offline etc.).
Compressão/Truncamento	– Permite inserir muito mais informação no prompt sem extrapolar limite. – Reduz custo de processamento removendo redundâncias do texto.	– Risco de perder detalhes relevantes se compressão for excessiva ou mal direcionada. – Pode requerer treino de um modelo auxiliar (p.ex., para sumarizar ou gerar *soft prompts*).	Variável: desde truncar heurístico (baixo) até treinar modelos compressores complexos (alto). Tecnologias modernas (xRAG, IC-Former) requerem pesquisa especializada.	**Sim.** Sumários e filtros podem ser feitos externamente. Integrações avançadas (embeddings como tokens especiais) exigiriam modificar a entrada ao `llama.cpp`, mas são factíveis com modelos open (ex.: inserir vetores como pseudo-tokens).

Artigos Experimentais e Estudos de Caso

Nesta seção, destacam-se alguns trabalhos técnicos que implementaram as técnicas acima em experimentos controlados, com **modelos open-source compatíveis com Llama.cpp** e **datasets abertos**, produzindo resultados reproduzíveis. Esses artigos podem servir de referência para projetos práticos, pois descrevem configurações de experimento, conjuntos de dados utilizados e melhorias obtidas com cada técnica de otimização de contexto.

- **RankRAG (Yue Yu et al., 2024)** – Este trabalho propõe um pipeline unificado de *retrieval + re-ranking + generation* utilizando um único LLM instruído para ambas as funções. Os autores finetunaram modelos LLaMA (apelidados de *Llama3* no artigo, com 8B e 70B de parâmetros) para que o próprio modelo pudesse **reordenar contextos recuperados e então gerar respostas**. Em avaliações de *open-domain QA* e conhecimento especializado, o RankRAG alcançou resultados impressionantes: um modelo de 8B/70B instruído com essa técnica **superou o GPT-4** em 9 conjuntos de dados de conhecimento geral, e obteve desempenho **equiparável ao GPT-4** em 5 conjuntos biomédicos específicos – tudo isso utilizando um modelo open-source bem menor. Isso demonstra o potencial de *fine-tuning* de LLMs abertos para internalizar etapas de recuperação e filtro de contexto. Além disso, o RankRAG mostrou que adicionar uma pequena quantidade de dados de ranking durante o treinamento melhorou tanto a capacidade de selecionar textos relevantes quanto a qualidade da geração final. Os experimentos cobriram datasets como **Natural Questions, TriviaQA, HotpotQA, 2WikiMultiQA** (QA aberto, incluindo *multi-hop*) e **FEVER** (verificação de fatos), todos abertos e bem conhecidos, o que reforça a reprodutibilidade. Esse artigo é um bom modelo para quem deseja replicar um sistema completo de RAG com modelos LLaMA ou similares.

- **Toolformer (Timo Schick et al., 2023)** – Este é um estudo pioneiro em *tool use* por LLMs. Os pesquisadores usaram um modelo GPT-J (6,7 bilhões de parâmetros, open-source) e o **treinaram para fazer chamadas a ferramentas externas** de forma autônoma. O treinamento foi auto-supervisionado: a partir de umas poucas demonstrações manuais de uso de APIs (como calculadora, busca web, tradução), o modelo gerou e aprendeu com seus próprios exemplos de quando usar as ferramentas. O resultado foi que o GPT-J afinado conseguiu **melhorar substancialmente seu desempenho em tarefas zero-shot**, chegando a resolver perguntas e cálculos com precisão próxima à de **modelos bem maiores (como GPT-3 de 175B)**, porém com uma fração do tamanho. Notavelmente, o Toolformer mostrou vantagens em tarefas como perguntas de conhecimento atual (usando busca online), cálculos aritméticos e tradução entre idiomas – áreas em que modelos puro-texto frequentemente falham. Este trabalho é relevante para experimentos pois disponibilizou uma metodologia de treinamento (que poderia ser reaplicada a outros modelos open-source compatíveis com *llama.cpp*, como LLaMA-7B ou Mistral-7B) para dotar o modelo de capacidades de ferramenta sem intervenção humana extensiva. Reproduzir algo semelhante requer implementar o loop de geração de chamadas e filtragem de quais chamadas reduzem a perda (como descrito no artigo), mas a recompensa é um sistema robusto capaz de interagir com o mundo exterior com alta precisão.

- **xRAG (Xin Cheng et al., 2024)** – Este artigo foca em **compressão extrema de contexto** para sistemas de *Retrieval-Augmented Generation*. A técnica xRAG introduzida pelos autores pega as embeddings densas dos documentos (geradas por um modelo de recuperação) e as **insere diretamente no espaço de representação do LLM**, através de um módulo conector treinável. Assim, ao invés de concatenar textos longos no prompt, o modelo recebe "pílulas" de informação codificada que ocupam pouquíssimos tokens. Nos experimentos, xRAG foi avaliado em cerca de 6 tarefas de conhecimento aberto (similarmente, NQ, TriviaQA e outras do gênero) e obteve em média **+10% de melhoria de performance** em comparação com baselines, igualando o desempenho de um modelo que lia todo o texto sem compressão em diversos *benchmarks*. Além disso, a compressão permitiu **reduzir o custo computacional em ~3,5 vezes**, agilizando a inferência. Os modelos de linguagem utilizados variaram de um LLM denso de 7B até um *Mixture-of-Experts* equivalente a 8×7B, todos architectures open-source (Transformers padrão). Isso indica que a técnica é compatível com modelos do porte do LLaMA-7B. Para quem deseja experimentar compressão de contexto, o xRAG fornece um caminho: preservar a modularidade (retriever separado) e apenas treinar uma ponte que alimenta embeddings ao LLM como substituto de texto. Embora implementar esse método exatamente possa exigir aprofundamento (e eventualmente modificar o código do modelo para aceitar essas representações fusionadas), os ganhos reportados sugerem que vale a pena investigá-lo para aplicações onde o contexto excede os limites tradicionais.

A tabela abaixo sumariza esses estudos de caso, enfatizando modelo, técnica, datasets e resultados principais obtidos:

Artigo (Ano)	Modelo(s)	Técnica(s)	Dataset(s)	Resultados-chave
RankRAG (Yu et al., 2024)	LLaMA 8B & 70B (fine-tuning instrucional)	Recuperação + Re-Ranking (unificados no LLM)	NQ, TriviaQA, HotpotQA, 2WikimQA, FEVER, etc. (9 tarefas gerais, 5 biomédicas)	LLM instruído **superou GPT-4** em 9 benchmarks de conhecimento geral e teve desempenho **comparável ao GPT-4** em 5 conjuntos biomédicos (zero-shot), demonstrando benefícios de integrar ranking ao modelo.
Toolformer (Schick et al., 2023)	GPT-J 6.7B (open-source)	Chamada de Ferramentas (auto-supervisionada)	Vários tasks abertos (QA, cálculo, tradução, etc.)	Habilitar uso de ferramentas **melhorou o desempenho zero-shot** do modelo significativamente, tornando-o muitas vezes **competitivo com modelos bem maiores** (e.g. aproxima ou supera GPT-3 175B em tarefas específicas), sem sacrificar habilidades linguísticas.
xRAG (Cheng et al., 2024)	LLM 7B (denso) e 8×7B MoE	Compressão de Contexto (fusão de embeddings no LLM)	6 tarefas de conhecimento (NQ, TriviaQA, etc.)	Método de compressão atingiu **~10% melhoria** média na exatidão e igualou a performance de modelos sem compressão em vários datasets, *reduzindo* os FLOPs de inferência em **3,5×**, comprovando eficiência sem perda de qualidade.

Sugestões de Datasets para Experimentos

Por fim, é importante selecionar datasets apropriados e abertos para conduzir experimentos de otimização de contexto com LLMs open-source. Abaixo estão algumas sugestões de conjuntos de dados *benchmark* amplamente utilizados, que possuem **disponibilidade pública** e podem ser empregados para reproduzir cenários como os dos artigos mencionados:

- **Natural Questions (NQ)** – Conjunto de perguntas reais do Google buscador, cada uma associada a trechos de Wikipedia como respostas. É um padrão ouro para QA de domínio aberto (*open-domain QA*), útil para testar recuperação de contexto e re-ranking, já que as respostas requerem buscar informações dispersas em artigos da Wikipédia.
- **TriviaQA** – Coleção de perguntas de trivia e conhecimento geral, com respostas baseadas em evidências textuais. Similar ao NQ, avalia a capacidade do sistema de encontrar fatos específicos. Possui versão *open* onde é necessário recuperar documentos pertinentes da Web.
- **HotpotQA** – Dataset de perguntas que exigem **raciocínio multi-hop**, ou seja, combinar informações de **múltiplos artigos** da Wikipedia. Cada pergunta vem com dois contextos distintos necessários para chegar à resposta. Excelente para testar fragmentação (já que os contextos são longos) e re-ranking (para encadear evidências corretamente).
- **2WikiMultiQA** – Outro dataset *multi-hop* que envolve perguntas cuja resposta requer verificar duas páginas da Wiki (versão multilingue). Útil para avaliar retrieval em cenários cross-artigo e até multilíngues.
- **FEVER** – **Fact Extraction and Verification**, um conjunto de afirmações que devem ser verificadas como Verdadeiras ou Falsas com base em evidências da Wikipédia. Serve para experimentos de verificação factual: requer recuperar frases específicas e possivelmente truncar contexto irrelevante, avaliando a capacidade do LLM de não alucinar e se basear apenas em evidência fornecida.
- **Doc2Dial** – Dataset de **QA conversacional** baseado em documentos do domínio de governo/serviços públicos. Cada diálogo se refere a um documento longo (por exemplo, um folheto informativo) do qual é preciso recuperar trechos para responder às perguntas do usuário ao longo do diálogo. Útil para testar recuperação em contexto de múltiplos turnos e *chunking* de documentos extensos.
- **TopiOCQA** – Conjunto de QA conversacional onde cada pergunta foca em um tópico específico e requer navegar um documento ou conjunto de documentos. Semelhante ao Doc2Dial, avalia ferramentas de contexto em conversas, e pode ser combinado com estratégias de memória de contexto (compressão ou seleção de turns relevantes).
- **KILT Benchmark** – Não é um dataset único, mas um **benchmark unificado** que abrange vários dos acima (NQ, HotpotQA, TriviaQA, FEVER, etc.). O KILT fornece divisão consistente de treino/validação/teste e uma infraestrutura para avaliação em tarefas de recuperação + geração. Usar o KILT pode facilitar a reprodutibilidade, pois já integra a consulta à Wikipédia e métricas padronizadas para checar se a resposta está sustentada pelas evidências corretas.

Cada um desses conjuntos de dados é compatível com experimentos em LLMs open-source, pois seus dados são públicos e geralmente já há *scripts* disponíveis para baixar e pré-processar (por exemplo, no `HuggingFace Datasets`). Além disso, todos contêm definições claras de tarefa e métricas objetivas (exatidão exata, F1, precisão de verificação, etc.), permitindo medir quantitativamente os ganhos das técnicas de otimização de contexto introduzidas – seja um aumento de precisão via melhor recuperação ou uma redução de alucinações via verificação factual. Ao escolher um dataset, leve em conta o foco do seu experimento: para avaliar **context retrieval e re-ranking**, QA abertas como NQ/TriviaQA são ideais; para **chunking e long context**, use HotpotQA ou diálogos de Doc2Dial; para **tool use**, tarefas especializadas (cálculo, tradução) podem ser montadas a partir de dados disponíveis (por exemplo, combinar questões matemáticas do dataset GSM8K para testar uso de calculadora). Com esses datasets e os artigos de referência, é possível conduzir experimentos robustos e **reprodutíveis** no contexto acadêmico ou de desenvolvimento de aplicações em LLMs, utilizando modelos open-source ajustados às necessidades de contexto do seu problema.