goal: instead of having the training of a large language model, we instead replace them with a series of small language models that are able to excel at a task (such as writing, citations, etc)

- Each SLM is an instruction-tuned model that is tuned for a narrow role and that is b) paired with a knowledge graph (a private KG) that is able to capture its domain/ entity

knowledge graphs + their roles: a global knowledge graph is able to hold canonical entities + relationships (people, paper, concepts, and style rules)

- each expert's local knowledge graph is a view/overlay with edges it trust most (sources, heuristics, and rating)
- periodic alignment merges +links nodes to global via entity linking and ontology mapping

inspiration and rationale

1. ***fedarated learning***

- We already know that federated learners are both computationally less expensive and thus less environmentally demanding

2. Knowledge graphs

- knowledge graphs are known to help with domain overlap

3. ***small language models:***

1. **economic and environmental advantages**

- **Inference** with small language models can be **10-30x cheaper than LLMs**; infrastructure cost for hosting monolithic models is disproportionately large-estimated at about 57 billion in 2024 vs the 5.6 billion dollar use in LLM api use; especially when we consider high volume agenitc applications, that cost scales massively

2. ***practically***

- they are more practical compared to general-purpose models, slms are faster and cheaper to run and require less memory and infrastructure, and are easier to deploy, especially in edge or on-device settings. Additionally, they are more agile; you can fine-tune them quickly and deploy updates without grinding your whole system to a halt.

3. ***hybridization, heterogeneity is ideal***

- Instead of relying on a general-purpose LLM, modularity is better. Slms handle routine, repetitive sub-task. Large LLMs are invoked for complex reasoning or in broad context understanding.; Having the right model for the right job approach reduces overhead and improves system flexibility

4. ***llms→slms***

- We can modularize LLMs to migrate them into slms

misc thoughts:

- LLMs are trying to do too much that's why we find them struggling so much
- Gpt-5 has improvements on technical tasks, but are weaker like writing; just like humans, instead of making one big human who will naturally fail at certain things, we should think as llms are a group of people, seeking the need for a federated approach.

architecture:

1. ***specialist SLMs ("experts")***

e.g., Writer, Editor, Fact-Checker, Citations, Policies/Style, Summarizer, Transcreator.

- Each SLM is (a) instruction-tuned for a narrow role and (b) paired with its private KG capturing its domain/entities.

2. ***KG fabric ("shared brain")***

A global KG holds canonical entities/relations (people, papers, concepts, style rules).

- Each expert's local KG is a view/overlay with edges it trusts most (its sources, heuristics, ratings).
- Periodic alignment merges/links local nodes to global via entity linking + ontology mapping.

3. ***Router + planner***

A tiny router SLM (or rules) classifies the subtask and picks the expert set.

- A planner (simple DAG executor) sequences experts: Draft → Fact-Check → Cite → Edit → Guardrail.

4. ***KG-aware retrieval + prompting***

Before an expert generates, it queries the KG(s) to pull entities, relations, and snippets.

- Prompts include a structured context block from the KG (facts, constraints, citations, style tags).

5. ***Aggregation & conflict resolution***

When experts disagree, run KG-guided arbitration: weigh claims by source reliability, recency, and graph centrality.

- Write outputs as frames (JSON with fields: thesis, key points, citations, risks). Merge frames, not free text.

6. ***Learning loop***

Log decisions, disagreements, and final picks as preference pairs to improve routers and experts (DPO/KTO).

- Promote new edges to the global KG when consistently corroborated.

how the KG pieces fit

Schemas:

Core ontology: Work -> cites -> Work, Claim -> supported_by -> Evidence, Entity -> related_to -> Entity,

StyleRule -> applies_to -> Section, Fact -> provenance -> Source.

- Keep it small + extensible (RDF/OWL or a property graph).

Storage

1. ***Property graph*** (Neo4j, Memgraph) for fast traversal + reasoning patterns.
2. ***Vector index*** (FAISS/pgvector) for semantic recall of entities/snippets.
3. Optional ***knowledge graph embeddings*** (TransE/RotatE/ComplEx) to score candidate links & overlaps.
4. ***Overlap discovery (your "domain overlap")***

- ***Entity linking***: same-as resolution by name, aliases, embedding cosine, and relation signatures.
- ***Graph alignment:*** match subgraphs via neighborhood similarity + label compatibility.
- ***Community detection:*** find topical clusters; use them to build "expert committees" dynamically.
- ***Confidence fusion:*** edge weights = f(source_reliability, recency, agreement_count).

expert lifecycle (per SLM)

1. ***Data curation***

- Small, clean instruction set (100k–2M tokens) + a few thousand gold examples with references.
- Auto-distill from a strong LLM, then hand-clean the top 10–20% you'll rely on.

2. ***QLoRA SFT***

- Train for the narrow role (e.g., Editor: clarity, concision, tone transforms).

3. ***Preference tuning***

- Collect 5–10k pairs: "Which output is clearer / more faithful / better cited?"

4. ***KG integration***

- Build a retrieval tool for the model (function-call/tool-use) that returns: entities, claims, citations.
- Teach the SLM to emit frames (JSON) and claim-provenance pairs for the KG to ingest.

inference flow (simplified)

1. ***Router:*** classify task → pick experts.

2. ***Planner:*** construct a pipeline DAG.

3. ***For each node (expert):***

- Query KG (global + local view) → get entities/claims/snippets.
- Generate frame (not raw prose).
- Write back proposed claims + provenance (tentative edges).

Aggregator:

- Merge frames with KG weighting; flag conflicts; ask Fact-Checker only where needed (cheap!).
- Produce final text from the merged frame via the Writer + Editor.

Log & learn:

Store disagreements, resolution rationale, user edits → future DPO + KG edge promotions.

concrete tech stack

Models: Llama-3.1-8B Instruct (general), Mistral-7B (writer), Phi-3-mini (router/tools), Qwen-7B (multilingual).
Training: HF Transformers + PEFT (QLoRA), DeepSpeed/FSDP, FlashAttention-2.
KG: Neo4j/ArangoDB; RDFLib if you prefer triples. Embeddings with OpenKE/PyKEEN.
RAG: LanceDB/pgvector + a slim GraphRAG layer (traversal → expand → rank → bundle).
Serving: vLLM + JSON schema-constrained decoding for frames; FastAPI orchestrator; Celery/Temporal for DAGs.
Eval: rubric scores (clarity, structure, faithfulness), citation coverage, hallucination rate, latency/cost dashboards.

aggregation strategies (pick one to start)

1. KG-routed MoE (simple & strong)

- Router picks N experts.
- Weighted voting on frames using KG trust scores.
- Only escalate to the big model if confidence < τ.

Plan-act-verify

- Writer drafts with KG context → Fact-Checker verifies claims via KG traversal → Editor enforces style.
- Loop only on flagged claims (cheap, targeted checks).

Dual-track (creative + literal)

- Creative track (Writer) + Literal track (Summarizer/Fact-Checker).
- Aggregator blends: Literal controls facts, Creative controls phrasing.

detecting/using domain overlap

Before routing: run entity extraction on the prompt, pull the induced subgraph around those entities, and see which experts have strong local edges there → pick committee.
During generation: experts propose new edges; aligner matches to global KG; if overlap grows across experts, boost those claims (consensus).
After delivery: store "topic → best committee" mappings for faster future routing.

risks & guardrails

1. ***Schema drift:*** keep the ontology versioned; add migrations.
2. ***KG bloat:*** enforce source quotas + evidence requirements to add edges.
3. ***Feedback loops:*** require cross-expert corroboration or external sources before promoting edges to global KG.
4. ***Latency***: cache frequent subgraphs; compress frames; limit committee size (usually 2–4 experts).

why this wins

1. ***Precision:*** experts stay in their lane; KG enforces factual spine.
2. ***Cost/latency:*** small models + targeted escalation only when uncertain.
3. ***Composability:*** you can add/retire experts like microservices.
4. ***Learning:*** every disagreement is training signal and a graph update.