Multi-Agent Architecture Analysis

Reactome LNP Agent — Model Strategy & Industry Comparison

Reactome LNP Agent v3.0

February 2026

LangGraph · AWS Bedrock · Claude Sonnet 4.5

This report analyzes the multi-agent architecture of the Reactome LNP Agent system, a 7-agent LangGraph pipeline for ionizable lipid design. It documents the current single-model configuration (Claude Sonnet 4.5 for all agents), compares it against industry-standard mixed-model approaches, and provides actionable recommendations for cost optimization, latency reduction, and architectural resilience.

1. Current Agent Architecture

The Reactome LNP Agent uses a **10-node LangGraph StateGraph** with 7 YAML-configured agents. All agents are powered by **Claude Sonnet 4.5** (`us.anthropic.claude-sonnet-4-5-20250929-v1:0`) via AWS Bedrock in **us-west-2**.

1.1 Pipeline Flow

The pipeline executes as a directed acyclic graph (DAG) with conditional branching based on query type:

Node	Agent	Role	LLM Instance	Temp
1	Query Rewrite	Rewrites question using chat history	llm_fast	0.0
2	Router	Classifies → synthesis / lookup / general	llm_fast	0.0
3	FAISS Retrieval + Rerank	454 vectors searched, LLM-based reranking	llm_fast	0.0
4-8	5 Expert Workers (parallel)	Domain-specific analysis	llm	0.1-0.2
9	Lead Agent	Supervisor — conflict resolution, final answer	llm	0.2

1.2 Agent Configuration Summary

Agent	Role	Max Tokens	Temp	Trigger	Type
Router	Query classifier	64	0.0	Always	Infrastructure
Reaction Expert	SMARTS template matching, feasibility	1,024	0.1	Synthesis	Chemistry
Lipid Design Expert	Retrosynthesis + SAR + design rules	2,048	0.15	Synthesis	Chemistry
Generative AI Expert	De novo generation + RL optimization	1,536	0.2	Synthesis	AI/ML
Property Prediction	ML models + uncertainty quantification	1,536	0.1	Synthesis	AI/ML
Literature Scout	PubMed, PubChem, web search	512	0.0	All paths	Search
Lead Agent	Supervisor — final answer authority	4,096	0.2	Always	Infrastructure

1.3 Model Configuration

Two LLM instances are created in `agent.py`, both pointing to the same Sonnet 4.5 model:

- **llm** — Default instance for expert workers and lead agent
- **llm_fast** — Same model, capped at 1,024 tokens for router, rewrite, and reranking

> **Key observation:** Every node in the pipeline hits the same Claude Sonnet 4.5 endpoint. The only variation is `max_tokens` and `temperature` per agent.

2. Industry Standard: Multi-Model Architectures

Production multi-agent systems in pharma, biotech, and AI research typically employ **tiered model assignment** — matching model capability to task complexity. This section surveys common patterns and their rationale.

2.1 Tiered Model Assignment

The industry consensus is to use the **cheapest model that meets quality requirements** for each agent role:

Agent Type	Typical Model	Rationale
Router / Classifier	Small/fast (Haiku, GPT-4o-mini)	Speed critical. 10-50ms matters. Simple classification.
Domain Experts	Strong reasoning (Sonnet, GPT-4o)	Needs domain knowledge + structured output.
Supervisor / Lead	Strongest available (Opus, o1, Sonnet)	Critical thinking, conflict resolution.
Retrieval Reranker	Embedding model or small LLM	Just scoring relevance — no generation needed.
Code / SMILES Gen	Code-specialized (Codex, DeepSeek)	Better at structured pattern generation.

2.2 Mixed-Provider Architectures

Some organizations go further and mix providers to optimize for different strengths:

Provider	Typical Use Case
Anthropic (Claude)	Complex reasoning, safety-critical analysis, long context
OpenAI (GPT-4o)	Code generation, function calling, structured output
Google (Gemini)	Multimodal analysis, very long context (1M+ tokens)
Open-source (Llama, Mistral)	Cost-sensitive tasks, on-premise requirements, fine-tuning
Specialized (ChemBERTa, MoLFormer)	Domain-specific tasks (SMILES validation, property prediction)

2.3 Real-World Examples

Pharmaceutical AI platforms (e.g., Insilico Medicine, Recursion):

- Use specialized chemistry models for molecular generation
- General-purpose LLMs only for natural language interface and report generation
- GNN/Transformer ensembles for property prediction (not LLM-based)

Enterprise AI agent frameworks (e.g., LangChain, CrewAI, AutoGen):

- Default recommendation: use smaller models for routing and tool selection
- Reserve large models for synthesis and final output generation
- Support model-per-agent configuration as a first-class feature

3. Comparative Analysis: Single-Model vs. Mixed-Model

3.1 Current Approach — Single Model (Sonnet 4.5 Everywhere)

Advantages

■ **Operational simplicity** — One API key, one provider, one billing stream. No cross-provider auth or version management.

■ **Consistent output style** — All agents produce similarly structured responses. The lead agent doesn't need to reconcile different formatting conventions.

■ **Simplified debugging** — Same model behavior everywhere. When output is wrong, the issue is always in the prompt, not model-specific quirks.

■ **Quality floor** — Sonnet 4.5 is genuinely capable enough for all tasks in this pipeline. No risk of a weak model degrading the chain.

■ **Prompt portability** — YAML prompts work identically across all nodes. No per-model prompt engineering required.

Disadvantages

■ **Cost inefficiency** — The router uses 64 output tokens for a 3-class classification. Haiku could do this at ~10× lower cost with equivalent accuracy.

■ **Latency overhead** — 5 parallel Sonnet calls have higher cold-start and TTFT than 5 parallel Haiku calls for simple tasks.

■ **Single point of failure** — If Bedrock throttles Sonnet 4.5 (rate limits, outage), the entire pipeline stops. No fallback path.

■ **No specialization** — A chemistry-fine-tuned model (e.g., ChemBERTa for SMILES validation) could outperform a general-purpose LLM on specific subtasks.

■ **Wasted capacity** — The reranker node just picks 4 indices from 8 candidates. This doesn't need a 200B+ parameter model.

3.2 Mixed-Model Approach (Industry Standard)

Advantages

■ **40-70% cost reduction** — Router + reranker on cheap models saves significantly. At scale (1000s of queries/day), this is substantial.

■ **Lower latency** — Router classification: ~100ms on Haiku vs ~500ms on Sonnet. This compounds across the pipeline.

■ **Provider redundancy** — If one provider has an outage, others still function. Critical for production SLA requirements.

■ **Task-optimized models** — Chemistry-tuned models for SMILES, code models for SMARTS patterns, embedding models for reranking.

■ **Scalability** — Different rate limits per model means higher aggregate throughput.

Disadvantages

■ **Operational complexity** — Multiple API keys, rate limits, billing streams, and version pinning across providers.

■ **Output inconsistency** — Different models structure responses differently. The lead agent must handle varied formatting and reasoning styles.

■ **Harder debugging** — 'Which model caused this bad output?' becomes a real question when 3+ models are in the pipeline.

■ **Per-model prompt engineering** — What works on Claude may fail on GPT-4o. Each model needs its own prompt tuning and testing.

■ **Integration testing burden** — Model updates from different providers happen independently, requiring continuous regression testing.

4. Side-by-Side Comparison

Dimension	Single Model	Mixed Model	Notes
Cost (per 1K queries)	$$$$	$$	Mixed model uses Haiku for 3 high-frequency nodes
Avg. Latency	~3-5s	~2-3s	Router + reranker 3-5× faster on small models
Debugging	Simple	Complex	Single model = one behavior to understand
Prompt Management	1 set	N sets	Each model may need different prompts
Provider Resilience	Low	High	Mixed model survives single-provider outage
Output Consistency	High	Medium	Same model = same style across agents
Quality Floor	High	Variable	Weak model in chain can degrade output
Specialization	None	High	Can use domain-specific models
Ops Complexity	Low	High	More providers = more moving parts
Scalability	Limited	High	Different rate limits = higher throughput

5. Recommendation

A **pragmatic middle ground** that stays within AWS Bedrock (no new providers) while optimizing cost and latency on high-frequency nodes:

Agent Group	Recommended Model	Rationale
Router / Rewrite / Reranker	Claude Haiku 3.5	Fast, cheap, sufficient for classification and ranking
5 Expert Workers	Claude Sonnet 4.5 (keep)	Strong reasoning needed for domain analysis
Lead Agent	Claude Sonnet 4.5 (keep)	Best reasoning for conflict resolution and synthesis

5.1 Expected Impact

Metric	Change	Notes
Router node cost	~80% reduction	64 tokens on Haiku vs Sonnet
Rewrite node cost	~80% reduction	Simple paraphrasing task
Reranker node cost	~80% reduction	Index selection, not generation
Pipeline latency	~20-30% reduction	3 fast nodes on Haiku
Quality impact	Negligible	Classification accuracy equivalent on Haiku
Implementation effort	~2 hours	Add model_id to YAML, update agent.py loader

Implementation: Add a `model_id` field to each agent YAML. Update `agent.py` to read per-agent model IDs and instantiate separate `ChatBedrock` clients. The router alone fires on every query — making it Haiku saves ~80% on that node with zero quality loss for a 3-class classification.

5.2 Future Considerations

- **Phase 2:** Add fallback model chains (Sonnet → Haiku) for resilience during throttling
- **Phase 3:** Evaluate chemistry-specific models (ChemBERTa, MoLFormer) for SMILES validation
- **Phase 4:** Consider Bedrock Guardrails for automated output validation across all agents
- **Phase 5:** Benchmark Haiku vs Sonnet on router accuracy with production query logs