ChatGPT

From Large Language Models to Small Models and Code: The Evolution of AI Solutions

By Dinis Cruz and ChatGPT (Deep Research)

Introduction

Large Language Models (LLMs) like GPT-3 and GPT-4 have revolutionized what machines can do with human language. These massive models (with hundreds of billions of parameters) stunned the world by exhibiting capabilities in understanding and generating text that were previously thought impossible. Crucially, LLMs served as *exploratory tools* – they allowed researchers and businesses to quickly **explore the art of the possible** in AI-driven applications, even if the initial solutions were inefficient or non-deterministic. However, as the field matures, we see a clear trajectory: **the era of AI as a "black box" is ending, with a shift toward smaller, more explainable models where determinism and provenance matter more than sheer size** [1]. In practice, this means moving beyond giant one-size-fits-all models to *Small Language Models (SLMs)* specialized for particular tasks, and ultimately to solutions embedded in **deterministic code or APIs**. This white paper charts that evolution – from LLMs to SLMs to code – and discusses why this transition is not only technologically logical but strategically advantageous for businesses and communities alike.

This evolution can be framed in terms of **Wardley Maps**, a strategic mapping technique that outlines how capabilities evolve from novel, chaotic solutions to stable, commoditized utilities. In the context of AI, we can map the journey of language processing capabilities as follows:

```
Genesis (Novel)  --->  Productization  --->  Commodity/Utility
[LLMs: Large, general models]  ->  [SLMs: Focused small models]  ->  [Code/
APIs: Deterministic solutions]
```

In the **genesis** stage, early large models act as trailblazers – powerful but inefficient and expensive. Through **productization**, we distill and refine these capabilities into smaller models tailored to specific needs. Finally, in the **commodity** phase, the functionality becomes standardized (often as software libraries or services), offering high reliability and low cost. This paper will delve into each stage, illustrating how organizations can leverage LLMs for innovation, then transition to efficient SLMs, and eventually to robust code or API implementations. We will also examine the competitive and security implications of this evolution, including how it aligns with trends like open-source AI and digital sovereignty. By understanding this progression, decision-makers can better allocate resources – using LLMs where they add unique value, but ultimately **codifying successful AI workflows into deterministic systems** for maximum efficiency and trustworthiness.

LLMs: The Exploratory Frontier

Large Language Models are characterized by their enormous scale and broad general-purpose training. The breakthrough of models like OpenAI's GPT series proved that with enough data and parameters, a single neural network can perform a wide variety of language tasks (from answering questions to writing code) without task-specific training. These LLMs operate as powerful generalists capable of behaviors that smaller models struggled with. In a sense, an LLM is like a heavyweight R&D lab encapsulated in a model – it can surprise us with novel solutions or insights. Organizations have eagerly adopted LLM-based solutions to do things that were previously impractical, such as generating human-like responses in chatbots, summarizing lengthy documents, or drafting software code from natural language prompts.

However, this potency comes with significant downsides. Large models demand extensive computational resources (often only runnable on cloud GPU clusters) and are **expensive** to operate at scale. They are also inherently **non-deterministic** – the same prompt may produce different outputs, and it's often impossible to fully explain *why* the model produced a given answer. In critical applications, this unpredictability (or "black box" nature) is a serious concern. Early adopters quickly noticed issues like *hallucinations* (confidently wrong answers) and difficulties in controlling the model's behavior. As an example, attempting to use a single giant model call to perform a complex task (like reading 50 news articles and picking the top five for a summary) revealed problems of explainability and repeatability: **How do we explain why those specific results were chosen? How can we ensure the same input yields the same output every time?** These were the challenges faced when an LLM tried to do too much in one step [2]. As one experimenter noted, when an LLM is tasked with a broad, multi-step reasoning job internally, we lose transparency into its process, making it hard to trust the results [2].

Despite these drawbacks, LLMs have been invaluable in the **exploration phase**. They demonstrated new capabilities – from creative writing to complex problem-solving – which established the *baseline of what is possible*. A historical analogy can be drawn to early inventions or feats: the first implementation might be ungainly or over-resourced, but it proves a concept can work. Once the capability is known, researchers and engineers naturally ask: *Can we achieve similar results more efficiently or reliably?* This question drives the evolution to the next phase: smaller, specialized models.

The Rise of Small Language Models (SLMs)

If LLMs are the broad explorers, **Small Language Models (SLMs)** are the focused specialists. An SLM is essentially a language model with far fewer parameters (typically in the millions to low billions) that is often **trained or fine-tuned for a specific domain or task** [3] [4]. While there is no hard cutoff, anything below about 10–30 billion parameters might be considered "small" in comparison to the 100B+ parameter giants [5] [6]. Crucially, what SLMs lack in sheer size, they make up in efficiency and relevance. By concentrating on narrower knowledge and vocabulary, a well-designed SLM can **match or even surpass a large model's performance on its target task** [7]. In fact, a recent survey of 160 research papers found multiple instances where models in the 1–8 billion parameter range performed as well as, or even outperformed, much larger models on specific benchmarks [7]. This is a testament to the power of specialization: a model doesn't need to know *everything* if it knows *exactly what's needed* for the job.

SLMs are made "small" and efficient through techniques like **knowledge distillation, pruning, and quantization**, which compress a larger model's knowledge into a leaner form [8] [9]. A classic example is **DistilBERT**, a distilled version of BERT (one of the early LLMs), which is about 40% smaller and 60% faster than the original while retaining **97% of BERT's language understanding capabilities** [10]. This illustrates that with careful training, we can dramatically shrink a model with only minor losses in accuracy. Additionally, fine-tuning allows an SLM to hone in on *domain-specific data*, giving it a depth of expertise that a general LLM may not have. For instance, a 3-billion-parameter legal model fine-tuned on court decisions can outperform a 100-billion-parameter general model when answering legal questions, simply because it "speaks the language" of that domain.

Benefits of SLMs. The move to small models brings a host of practical benefits:

- **Lower Compute and Cost:** SLMs require far less memory and processing power, which means they can run on commodity hardware (even laptops or smartphones) rather than expensive cloud GPU rigs [11]. This drastically lowers the cost of deploying AI – both in terms of cloud bills and energy consumption – making AI features more accessible to startups, open-source projects, and even on-device applications. Deployment becomes cheaper and greener, reducing barriers to entry [12].

- **Faster and Real-Time Performance:** With fewer parameters to churn through, SLMs generally have quicker inference times. This enables snappier real-time interactions – a critical factor for user-facing applications like customer service chatbots or mobile assistants. Users increasingly expect instant responses, and smaller models are better suited to meet that expectation.

- **On-Device Processing & Privacy:** Unlike giant models that almost always reside on cloud servers, many SLMs can be deployed **at the edge** – on personal devices or local servers – without an internet connection [13]. Projects like *llama.cpp* have famously demonstrated that even a consumer laptop or smartphone can run a moderately capable language model completely offline [14]. Edge deployment enhances privacy and security: sensitive data never leaves the device, and users are not at the mercy of a remote service. For organizations concerned with data sovereignty or compliance (consider stringent EU data protection rules), this is a huge advantage.

- **Customizability and Domain Expertise:** SLMs are easier to fine-tune on niche datasets, allowing the creation of models that speak the jargon of a particular industry or handle tasks in a constrained context [15] [4]. This specialization often leads to **more accurate and relevant results within that domain** – fewer irrelevant outputs, fewer hallucinations on in-domain queries, and often reduced bias, since the training data can be more tightly controlled [16]. In other words, an SLM can be *made to measure* for a problem, whereas an LLM is a one-size-fits-all suit. As a bonus, the smaller size and focused nature of SLMs also mean **fewer unwanted capabilities** – which translates to a smaller attack surface and less risk of the model being exploited or misbehaving outside its intended scope [17] [18].

- **Determinism (to an extent):** While SLMs are still probabilistic models at their core, their narrower scope can make their behavior more predictable in practice. With a limited palette of knowledge and responses, an SLM's output variability is constrained by design. This is beneficial for tasks where consistency is more important than open-ended creativity.

Of course, SLMs have **limitations** too. By focusing on specific tasks, they sacrifice the broad knowledge of an LLM – if taken out of their domain, they may fail spectacularly or produce irrelevant answers [19]. Training or fine-tuning many small models for different tasks requires effort and maintenance of multiple systems rather than one central system. There is also an inherent trade-off between **generality and determinism**: the more we constrain a model, the less likely it will handle unforeseen queries. However, in many business scenarios, *we don't need a model that can do everything – just one that can do the required thing efficiently and safely*. This realization is driving a wave of commoditization in AI: tasks that once needed a cutting-edge LLM can now be accomplished with a lean model or even a bit of classical code. Nowhere is this commoditization clearer than when we apply a **Wardley Mapping** lens to the AI ecosystem.

Wardley Mapping the Evolution of AI Capabilities

Wardley Maps illustrate how technological capabilities evolve from novel experiments to standardized commodities. Applying this to AI, we see LLMs and SLMs not as competitors but as **different evolutionary stages** of similar capabilities. Initially, only a few actors (typically large tech companies or research labs) could deploy LLMs – these were the **Genesis or Custom-Built** phase: bespoke, expensive, and accessible to a select few. Over time, as the knowledge of "what's possible" spreads and tools improve, similar capabilities move into a **Product or Productized** phase – more affordable, more common, and packaged in smaller, specialized offerings (this is where SLMs come in). Finally, the capability becomes a **Commodity or Utility**, available on-demand as a standardized service or as open-source code that anyone can run.

In the context of language AI, the **user need** might be something like "convert unstructured information into useful answers or actions." In 2020, the only way to meet that need might have been an LLM accessible via a limited API (a high-end, costly service). By 2025, we have open-source models and small fine-tuned models that can fulfill many such needs running locally or at a fraction of the cost. We are headed toward a future where asking a computer to summarize a document or answer a question is as routine and commoditized as doing a web search or a database query is today.

From a strategic business perspective, this evolution has profound implications. The **incumbents** (big tech firms that poured massive investment into giant models and control access to them) naturally want to keep AI capabilities in the realm of high-tech *"magic"* that only they can provide. Their cloud AI APIs, based on proprietary LLMs, are a lucrative product. However, the Wardley Map suggests an inevitable pull towards **commoditization**: what starts as custom and novel eventually becomes standardized and cheap (or even free). Indeed, we are already seeing this: **the AI community is rapidly commoditizing what used to be the sole domain of trillion-parameter models, by using clever distillation and open-source collaboration to create smaller, efficient models available to all** [7]. The barriers to entry are falling. For example, the open release of models like Meta's LLaMA (and its successors) to researchers sparked a proliferation of specialized models that rival the capabilities of closed models, but which can be run on ordinary hardware by anyone. This open-source movement is pushing AI along the evolution curve faster than many anticipated.

Europe offers a case study in how these trends align with regional strategy. European policymakers and researchers have recognized that the *"next phase of AI evolution aligns with Europe's values and capabilities"* [20]. Specifically, the emerging emphasis on **transparency, determinism, sustainability, and decentralization** plays to Europe's strengths [21] [1]. EU regulators, for instance, are pushing for AI systems that can explain their decisions and provide provenance for their information – requirements that massive opaque LLMs struggle with, but which smaller models or symbolic AI can better accommodate [1]. Additionally, Europe's pursuit of "digital sovereignty" favors solutions that can be locally hosted and controlled, rather than relying on foreign-owned cloud services [14]. SLMs running on edge devices or domestic servers fit well here. European AI experts argue that the **future of AI is more federated and open**, as opposed to dominated by a handful of Silicon Valley corporations [22] [23]. In fact, a recent deep-dive identified *six defining trends* that underpin a strategic opportunity for Europe, including the rise of commoditized models and deterministic AI, the shift from cloud to edge, and the primacy of open-source and trustworthiness [1] [22]. These trends can be summarized as a broader industry shift that rewards a different set of priorities than the last AI wave did:

- **Quality over quantity** – prioritizing smaller but more *reliable* models, rather than judging by parameter count alone [23].

- **Specialization over generalization** – leveraging **specialised domain expertise** instead of one-size-fits-all systems [23] .
- **Federated innovation over monopolies** – favoring collaboration, open standards, and distributed development rather than concentration in a single platform or provider [24] .
- **Trust and ethics as differentiators** – viewing transparency, fairness, and human-centric design as competitive advantages, not compliance burdens [25] .

These principles echo the Wardley progression: as the basic capability (e.g. language understanding) becomes ubiquitous, competitive advantage shifts to *how* you apply it (with quality, trust, and domain fit) rather than *who* has the biggest model. A useful analogy is the evolution of computing hardware – in the mainframe era, only IBM-sized players could play, but today computing power is commodity and value is created by how you use it (software, user experience, integration). Likewise, large general AI models are on the path to commoditization; the value will increasingly lie in bespoke fine-tuning, clever integration, and the last-mile delivery of AI to solve real problems.

From the standpoint of **Wardley Mapping** strategy, organizations (and regions like Europe) that anticipate this evolution can position themselves to benefit. Embracing open-source models, contributing to common AI frameworks, and building expertise in adapting models to local needs can ensure they remain competitive when AI capabilities are ubiquitous. Conversely, clinging to proprietary giants as a moat may prove short-sighted – history shows that such moats are eventually breached by the tide of commoditization. The smart play is to skate to where the puck *is going*: in AI, that puck is clearly sliding toward smaller, specialized, and standardized solutions.

From SLMs to Code: The Path to Determinism

An even more deterministic and cost-effective step beyond specialized models is to **remove the model entirely** from parts of the workflow. In other words, once you thoroughly understand a task that an AI is performing, you can often implement that task with traditional software – algorithms and code – achieving results that are faster, perfectly repeatable, and easier to verify. Code is, by its nature, **explicit and deterministic**: given the same input, it will produce the same output every time (barring bugs). It doesn't hallucinate, and its logic can be inspected and tested. Therefore, a key part of the LLM → SLM → Code evolution is knowing when to **promote** an AI capability into a hard-coded or rule-based solution, or into an API call to a well-defined service.

This progression is already happening in various domains. Consider the earlier example of using an LLM to sift through news articles for a personalized briefing. The initial LLM approach struggled with transparency: you'd get 5 articles, but *why those 5?* To address that, researchers turned the problem into a series of smaller, manageable steps and introduced a **semantic knowledge graph** to explicitly track relationships between topics, articles, and a user's interests [26] . In that solution, the LLM was used in early phases to extract entities and connections (essentially helping build the graph), but the later phases – like finding which articles connect most strongly to a user profile – could be done by straightforward graph traversal and scoring algorithms [27] [28] . The important insight was that *once the data was structured in a graph, an algorithm could take over the reasoning*. In fact, the architect of that system noted that one of the LLM-heavy steps "**should eventually not be needed** once we provide a strong ontology and taxonomy… we would be able to do the same by just navigating the graphs" [29] . In other words, the goal is to **use the LLM to figure out the solution, then use code to execute the solution** thereafter.

This pattern is broadly applicable: use LLMs and SLMs as **intelligent automators** to prototype and discover approaches, then consolidate those approaches into deterministic code or database queries. For example:

- In **customer support**, an LLM might observe patterns in how issues are resolved and suggest an algorithm for routing tickets or a set of if-then rules for common problems. Those rules can be implemented in code, eliminating the need to consult the LLM for every ticket triage.

- In **content filtering**, an LLM could be used initially to label a large dataset of content as acceptable or not, possibly uncovering complex cues. But if those cues can be distilled into a set of keywords, regex patterns, or a smaller classification model, the real-time filtering can be handed off to a deterministic system that is faster and cannot "change its mind" unpredictably. This is critical for performance and legal compliance – you want your filter to behave consistently over time.

- In data analytics or **report generation**, you might start by having an LLM generate narratives from data. By studying its outputs, you could derive a template or formula that covers 90% of the cases. Then a simple script or an API call can produce those narratives automatically, calling on an LLM only as a fallback for truly unusual cases.

There is also a middle ground in the form of **Retrieval-Augmented Generation (RAG)** and similar architectures. RAG employs a static knowledge store (e.g. a vector database or knowledge graph) in conjunction with a model. The model's job is reduced to understanding the query and generating a final answer, while the heavy lifting of storing and retrieving facts is done by the knowledge base. This approach already moves us closer to determinism: the answer is constrained by factual data retrieved, and the model is less likely to wander off-topic or hallucinate. Still, as Dinis Cruz observes, even vector databases and embedding-based retrieval have a bit of "black magic" in them – they introduce their own parameters (embedding dimensions, similarity thresholds) that need tuning. The aspiration is to keep pushing more of the process into the fully explainable realm. Ideally, if the knowledge is structured well (say, in a graph or relational DB), a user query could be answered by a deterministic query or algorithm, *no language model needed at all*. Achieving that ideal in complex domains is an ongoing challenge, but every step in that direction yields improvements in **speed, cost, and trustworthiness**.

One should not misconstrue this as "LLMs will be obsolete." On the contrary, LLMs (especially large ones) remain extremely useful for what they uniquely offer: **flexibility and adaptability in unfamiliar situations**. They are great at prototyping, at bridging gaps where formalized logic is not yet available, and at providing intuitive outputs that spark new solutions. The point is to **use LLMs deliberately and sparingly** – harness them to discover the solution, then solidify that solution into code. Over time, the role of LLMs may shift to more of a *teaching and meta-level*, where they help us write software or design models, rather than being deployed for every end-user query.

Security and Governance Considerations

An important yet sometimes overlooked benefit of this evolution toward smaller models and code is **improved security and governance**. Large generalist models come with significant security challenges. By virtue of being trained on vast swathes of the internet, they can output almost anything – including sensitive information or problematic content – if prompted the right (or wrong) way. They also present a broad **attack surface**: malicious actors can attempt prompt injection attacks, manipulate the model into revealing proprietary data from training, or coerce it into performing harmful actions.

Controlling a system that is essentially a black box with an almost unlimited knowledge base is very difficult.

Small models and deterministic systems inherently reduce these risks. When a model is domain-specific and limited in what it knows and can do, the opportunities for misuse shrink. As one industry guide notes, having fewer parameters and a narrower focus makes a model “more contained, providing a smaller attack surface” and easier to secure [17] [18]. For example, a customer-service chatbot SLM fine-tuned only on a company’s support FAQ will be far less likely to produce an inappropriate response or leak information than a giant uncensored model with all of Wikipedia in its memory. In addition, deploying models on-premises or on-device (which SLMs enable) keeps data under the organization’s control [18]. This mitigates risks associated with sending data to third-party cloud APIs and is crucial for compliance with data residency laws. Many companies are already leery of using external LLM APIs for anything but the most generic tasks, precisely because of data governance concerns. Running a self-contained SLM or using a deterministic code path gives them back control.

When a process is translated fully into **code or fixed rules**, the security improves further: one can apply traditional software security audits, threat modeling, and patch management. Engineers can reason about all possible inputs and outputs (something not feasible with an AI model that might generalize in unpredictable ways). Moreover, deterministic systems can be subjected to formal verification or at least rigorous unit and integration testing. None of these techniques are straightforward with a large neural network whose internal logic is opaque. In regulated industries (finance, healthcare, aviation, etc.), the ability to *explain and guarantee* system behavior is often not just a preference but a legal requirement. We foresee that AI features in such safety-critical or compliance-centric fields will increasingly be implemented as deterministic algorithms that were *informed by AI during design*, rather than live LLM-driven processes.

Finally, the security advantage ties back to the Wardley map discussion: a smaller attack surface and easier governance are competitive advantages. Companies and governments that prioritize them can deploy AI more broadly and with greater confidence. For example, an open-source ecosystem of small models can be inspected for backdoors or bias, in contrast to proprietary LLMs where one must trust the provider. This transparency builds **trust** – users and oversight bodies are more likely to accept AI when they know it’s not a mysterious black box influencing outcomes [1]. In an era of increasing concern about AI safety, being able to demonstrate that your AI is understandable and controllable could become a market differentiator.

Strategic and Business Implications

The evolution from LLMs to SLMs to code carries strategic lessons for both businesses and policymakers:

- **Innovation to Efficiency Pipeline:** Organizations should structure their AI efforts as an innovation pipeline. In the early phase, allow data scientists to experiment with large general models to quickly prototype new capabilities – this is where you maximize creativity and discovery. But once a prototype shows value, invest in **distilling that solution**. This might mean fine-tuning a smaller model for just that task, or rewriting the solution as explicit code. By doing so, you move from a high variable cost (paying per API call or per GPU-hour for an LLM) to a low fixed cost (running a small model or code at will). The result is often a huge boost in ROI for the AI feature: the service becomes faster and cheaper to run, enabling you to scale it to more users or incorporate it more deeply into your products.

- **Reduced Dependence on AI Monopolies:** Relying solely on the API of a big LLM provider can create lock-in and expose you to supply and pricing risks. As smaller models and open-source alternatives mature, businesses have leverage to avoid being bottlenecked by the roadmap or pricing of a single vendor. We've seen how quickly the open AI community can replicate proprietary capabilities – from image generation to language chat – and often at a fraction of the cost. Adopting an open or hybrid approach (where you use big models only for what truly requires them, and otherwise use your own SLMs or code) can be both cost-effective and strategically sovereign. This is precisely the approach championed in proposals for an **open-source sovereign cloud** in Europe: by building 100% open-source AI infrastructure, European industries could ensure long-term sustainability and interoperability, rather than renting intelligence from abroad [30] [31]. The same logic applies to companies – owning your "AI stack" (even if composed of smaller models and classical algorithms) can be more sustainable than outsourcing your core competencies to an AI API.

- **Human Capital and Skills:** As AI systems transition to code, the skill sets needed in your team also shift. Early on, you might need prompt engineers and AI researchers to wrangle LLMs. Later, you'll need software engineers who can interpret AI outputs and integrate them into production systems. There is a convergence happening: software developers are learning to use AI as a tool (for example, using LLMs to generate code snippets or unit tests), and data scientists are learning the constraints of production software (like efficiency, maintainability, security). Cultivating a team that is fluent in both AI and traditional engineering will pay off. They will know when to call the magic of an AI model and when to instead write a deterministic function. They will also be able to maintain the hybrid systems that result (e.g. an application that uses a mixture of model queries and database queries).

- **Differentiation through Trust and Experience:** As baseline AI capabilities become ubiquitous, competitive differentiation will come from offering superior trust, personalization, and user experience. If all your competitors can also summarize texts and generate code with commodity models, you win by doing it *better* – perhaps by ensuring your system never fails the user in obvious ways, by respecting privacy, or by integrating seamlessly into a user's workflow. This again points to the importance of determinism and specialization. A tailored small model or a piece of code might not have the flashiness of GPT-4, but if it **never gets facts wrong in its narrow domain**, users will prefer it for that reliability. Businesses should ask: where is it acceptable to have a "pretty good most of the time" AI, and where do we need a guarantee of correctness or consistency? Allocate LLMs to the former and code to the latter.

- **Regulatory Preparedness:** Governments are increasingly interested in regulating AI outputs (for misinformation, bias, copyright, etc.). Having more of your AI logic in transparent code or smaller models makes compliance easier. You can audit and tweak these systems to ensure they meet standards. If you deploy a black-box LLM and it produces a problematic result, you might have little recourse or ability to fix it beyond filing a bug report to the provider. In contrast, if an issue arises in a deterministic component, your engineers can typically diagnose and patch it. This agility in response is important in a fast-evolving regulatory landscape.

Conclusion

The journey from LLMs to SLMs to code represents the natural maturation of AI technology. We began with awe-inspiring generalists that showed us what's possible – large language models that could carry on conversations, write software, and score high on exams. But just as every major technology transitions from its **age of discovery** to an **age of utility**, AI is now moving toward solutions that are optimized, reliable, and integrated into everyday tools. **Large models opened the door; small models and code are walking through it.**

In practical terms, this means the future of AI is not one mega-model to rule them all, but rather a constellation of smaller models, micro-models, and conventional software working in concert. For developers and businesses, the mandate is clear: *use the right tool for each job*. Harness LLMs for their breadth and creativity when you're facing a blank slate or a novel challenge. But as the problem and solution crystallize, encapsulate that knowledge into leaner models or explicit code. Doing so brings numerous benefits – speed, cost savings, explainability, and security – which ultimately translate into better products and services.

Strategically, the shift to smaller models and open algorithms democratizes AI. It lowers the entry barrier for new players and empowers regions or organizations that weren't ahead in the race for the biggest model. An open, commodity foundation of AI means everyone can build upon it, much like open-source operating systems or the internet protocols. The value creation then moves "up the stack" – to the unique data you have, the user experience you design, and the specific problems you solve. In this emerging paradigm, **European initiatives focusing on open-source, multilingual and culturally aware AI are poised to thrive, as are any efforts that emphasize human-centric and trustworthy AI practices** [32] [1] .

In conclusion, the evolution from LLM to SLM to code is a story of *AI growing up*. We are preserving the magic that these models introduced, but grounding it in solid engineering and ethics. The result will be AI systems that are not only powerful, but also **efficient, transparent, and dependable**. Organizations that recognize this trajectory and adapt accordingly – combining the exploratory mindset of using LLMs with the rigor of engineering deterministic solutions – will lead the way in the next chapter of the AI revolution. The message is clear: the future of AI will not be about owning the biggest model, but about **how wisely and seamlessly you can integrate intelligent components of all sizes into solutions that truly benefit people**.

---

[1] [14] [20] [21] [22] [23] [24] [25] [32] Europe's Strategic Opportunity in GenAI: Why the Future of AI Plays to European Strengths
https://www.linkedin.com/pulse/europes-strategic-opportunity-genai-why-future-ai-plays-dinis-cruz-fp8me?trk=public_post

[2] [26] [27] [28] [29] Building Semantic Knowledge Graphs with LLMs: Inside MyFeeds.ai's Multi-Phase Architecture
https://mvp.myfeeds.ai/building-semantic-knowledge-graphs-with-llms-inside-myfeeds-ais-multi-phase-architecture/

[3] [4] [6] [16] [17] [18] Small language models: A beginner's guide | Ataccama
https://www.ataccama.com/blog/small-language-models

[5] [8] [9] [11] [12] [13] [15] [19] Small Language Models (SLM): A Comprehensive Overview
https://huggingface.co/blog/jjokah/small-language-model

[7] Small Language Models (SLMs) Can Still Pack a Punch: A survey
https://arxiv.org/html/2501.05465v1

[10] What are Small Language Models (SLM)? | IBM
https://www.ibm.com/think/topics/small-language-models

[30] [31] PDF: An Open-Source Sovereign Cloud for an Open Europe: The Case for a Federated, AI-Enabled, and Multilingual Digital Infrastructure | Dinis Cruz
https://www.linkedin.com/posts/diniscruz_an-open-source-sovereign-cloud-for-an-open-activity-7299764713243004928-6fhv