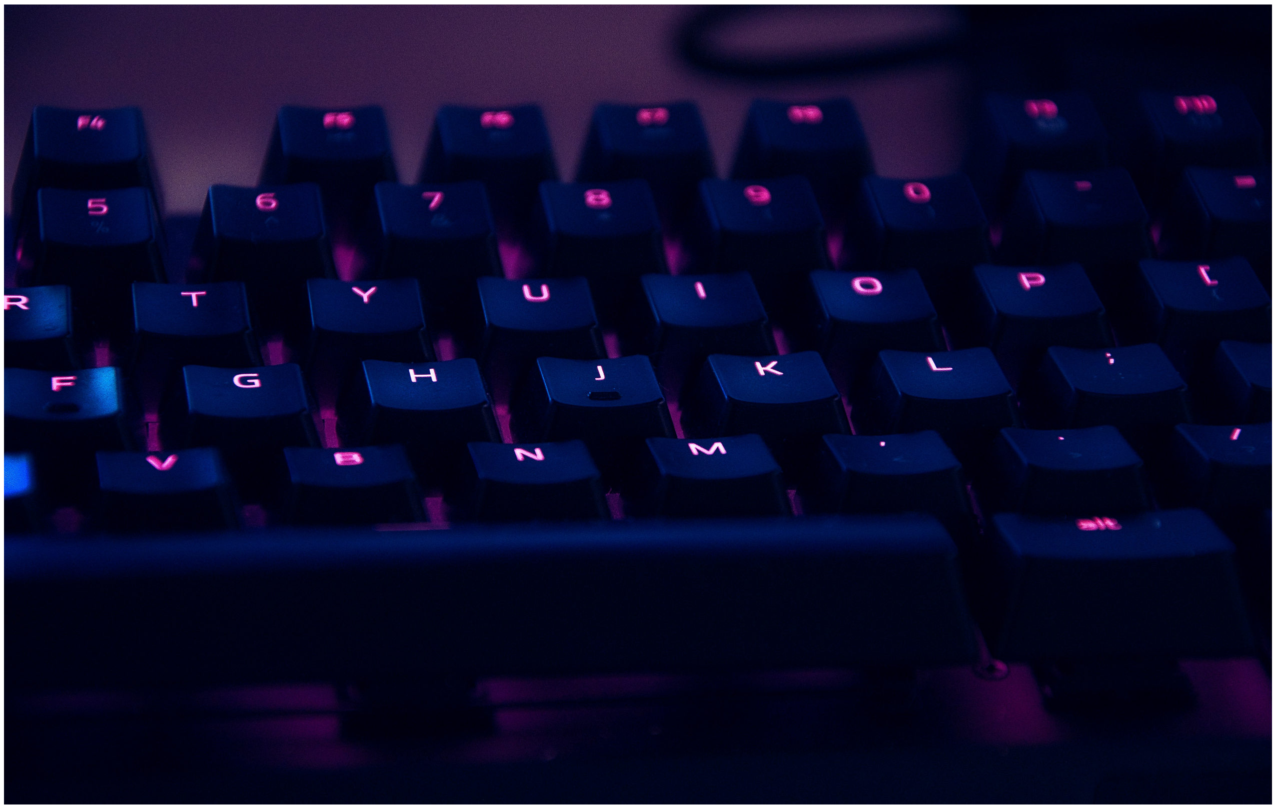
Building a Voice-Enabled Mortgage Assistant in Four Hours: A Lesson in Compound AI Systems

AI, Leadership | Aug 14, 2025



Most groundbreaking banking technology stories start with months of planning and end with a cautious pilot. This one starts with a single "what if" idea and ends four hours later with a working prototype in hand.

We built a fully functional voice-enabled mortgage application system in a single afternoon. It works. It processes natural speech, understands context, handles ambiguity and produces structured loan applications. Total development time: four hours. Total cost per conversation: less than a penny.

Then we killed it.

This is the story of how we chained together modern AI tools to create something genuinely useful — and why knowing when to stop is just as important as knowing how to start.

The Power of Compound AI Systems

At Nomis, we've spent two decades building analytics and workflow solutions for banks. Our Nomis Deal Manager (NDM) product is widely used across Canadian financial institutions, helping frontline employees navigate complex mortgage conversations. It's a proven, deeply integrated, text-based workflow tool that captures every detail necessary for processing applications. While functional, though, it is also an inevitable bottleneck in getting mortgages closed and on the books. A staff member at the bank must sit between the customer and the final approval, manually changing structure and pricing. We'd like to find a faster path to that decision.

So, our leadership team asked a simple question: "What if customers could just talk to it?"

The vision was compelling. Transform NDM from a desktop application into a conversational interface accessible via mobile app or phone call. Let users describe their needs naturally, in their own words, while our system handles the complexity behind the scenes.

Here's what makes this project interesting — we didn't build a single AI model. We orchestrated multiple specialized tools, stitching together a functional workflow with a combination of existing technology and our deep domain knowledge. Each tool excels at one thing. Together, they create something none could achieve alone.

The Technical Architecture

Our stack was deliberately simple:

- **Whisper** (running locally): Three models for speech-to-text comparison
- **GPT-3.5**: Context-aware response parsing

Total external dependencies: One OpenAI API key. Everything else runs locally, including the 500MB of Whisper models that power the speech recognition.

Five Engineering Decisions That Made It Work

1. Natural Conversation Through Voice Activity Detection

Rather than implementing push-to-talk or fixed recording windows, we built sophisticated voice activity detection:

```python
if volume > SILENCE_THRESHOLD:
    is_speaking = True
    silence_start = None
elif is_speaking and (time.time() - silence_start > SILENCE_DURATION):
    # Stop after 2 seconds of silence
    break
```

Two seconds of silence feels conversational, not robotic. Users speak naturally, pause to think, and continue — just like real conversation. The system waits patiently, then processes when they're done.

2. Performance Transparency with Production Pragmatism

Our first instinct was to use Whisper's largest, most accurate model. Twenty to forty seconds later, we were still waiting for results. That's when we learned an important lesson: perfect accuracy is worthless if users abandon the conversation.

We quickly pivoted to testing all three smaller models simultaneously:

```python
for model_name in ["small", "base", "tiny"]:
    start_time = time.time()
    result = model_data["model"].transcribe(audio_array)
    transcription_time = time.time() - start_time

    if model_name == "base":
        print(f"({transcription_time:.2f} sec. WILL USE THIS ONE)")
```

Small model: 2-5 seconds, highest accuracy
Base model: 0.5-2 seconds, optimal balance ✓
Tiny model: 0.2-0.8 seconds, fastest

The base model hit the sweet spot — fast enough to feel responsive, accurate enough to be reliable. During development, running all three simultaneously gave us real-time performance

Instead of writing complex regex patterns for every possible way someone might express a dollar amount, we let GPT handle the ambiguity:

```python
prompt = f"""
The user said: "{user_text}"
Extract the numeric value of the house.
Handle formats like "$X", "X dollars", "X million", "Xmm", etc.
Return ONLY the numeric value.
"""
```

Someone says "one point five mil"? GPT understands. "1.5 million dollars"? Got it. "$1,500,000"? No problem. We maintain a regex fallback, but GPT handles 95% of cases without breaking a sweat.

4. Platform Agility Over Platform Purity

We initially used PyAudio, the standard Python audio library. It crashed on M1 Macs. Within 30 minutes, we pivoted to sounddevice:

```python
import sounddevice as sd  # Works everywhere
```

No ego, no attachment to "the right way." The best tool is the one that actually works on your target platforms. This pragmatism saved hours of debugging and enabled team members on different hardware to contribute immediately.

5. Structured Output from Unstructured Input

Voice conversations are messy. Business processes need structure. We bridge this gap elegantly:

```python
loan_data = {
    "call_datetime": datetime.now(),
    "name": "",
    "state": "",
    "house_value": 0,
    "loan_amount": 0,
    "offered_rate": 0,
    # ... complete structured data
}

# Natural conversation happens here...

# Then produce pristine output:
print(tabulate(table_data, tablefmt="grid"))
```

Why We Killed It (And Why That's Good)

From idea to functional prototype in 4 hours? Amazing! But, this, in our minds, is where the use of AI gets interesting. Just because we *can* build it, does that mean we *should*? The constraint in software has always been engineering capacity. We have far more ideas than we have fingers on keyboards, so we have to make hard trade-off decisions on a daily basis. Does AI change that equation? Not really. Not yet, anyway.

We had a working prototype. It could conduct complete mortgage application conversations, handle rate negotiations and produce structured documentation. But, as any development team knows, the difference between a working prototype and a tested, production ready, enterprise grade solution can be vast. It feels like a working prototype is 90% of the way done. But that last 10% tends to be what takes far longer than our estimates ever allow for.

In our case, deal manager only works when data flows smoothly to both upstream and downstream systems. All of the corner cases need specialized workflows with hierarchical approval chains. The allowable inputs need error handling, and must incorporate regulatory guidelines as well as policy limits. And the real kicker? We work with large, global financial institutions, which means that every single customer will have unique and highly specialized requirements in those areas that are specific to that bank, and that bank alone.

Our last 10% was going to require extensive API updates, connecting to our complex configuration layer, incorporating multiple composable APIs, and new rounds of UAT with each individual customer. This is months of work, even with our trusty coding agents along for the ride. We did our value versus effort evaluations, and the answer was clear. Yes, the effort was dramatically reduced by using AI. But, the product just doesn't pencil out in terms of value to either Nomis or our customers to justify the remaining effort.

This is mature product development. Not every prototype becomes a product. The ability to build something quickly, evaluate it honestly and walk away when appropriate is as valuable as the technical skills themselves. And while AI will allow us to evaluate exponentially more ideas in a shorter time frame, it doesn't change the fundamental fact that we still have tradeoff decisions to make. If anything, it makes these decisions more important than ever. In our opinion, the teams that can best navigate this balance between agility and discipline will win in the new AI powered technology race.

The Real Lesson: Compound Systems Over Individual Tools

The AI landscape is littered with single-purpose tools that promise revolution. Chat interfaces that can't integrate. Voice systems that can't handle context. Analytics that can't communicate.

This compound approach — chaining specialized tools rather than waiting for one perfect solution — is how we're embedding AI across our entire platform. Our analytics agents don't replace human judgment, they accelerate it. Our optimization tools don't eliminate expertise, they amplify it.

What's Next

While this particular prototype won't see production, the lessons learned are influencing our product development. We're applying the same compound AI approach to enhance our core NDM platform, our pricing optimization tools and our analytics suite.

The code for this prototype is available on our GitHub. It's rough — remember, four hours total— but it's real, it works and it demonstrates what's possible when you stop thinking about AI as a single tool and start thinking about it as a toolkit.

Explore the full GibHub Repository here.

Ready to see how compound AI systems can transform your analytics capabilities?

Contact us at sales@nomissolutions.com to explore how we're applying these principles to solve real banking challenges.

Written by: Wes West, Chief Analytics Officer at Nomis Solutions

↖ BACK TO BLOG