OpenAI Responses API Report

What is Responses API? How should we use it?

1. **Definition**:

- A newer OpenAI API that provides more structured and reliable responses
- Ensures responses adhere to specified JSON schemas
- Provides explicit handling for refusals and errors

2. **Key Features**:

- Schema enforcement for consistent response formats
- Better error handling with explicit refusals

- Streaming support for real-time responses
- Context management for conversation history

3. **How to Use It**:

- Define JSON schemas for each response type
- Implement schema-based responses for critical methods
- Handle refusals and edge cases programmatically
- Gradually adopt across your codebase

4. **Implementation Strategy**:

- Start with one method (e.g., `GenerateTitleAssistant`)
- Define schemas for structured data (titles, dimensions, ratings)
- Update parsing logic to handle schema-based responses
- Test with real-world examples before broader adoption

What is Agents SDK? How should we use it?

1. **Definition**:

- Framework for creating AI-powered autonomous agents
- Allows agents to make decisions and take actions
- Enables coordination between multiple agents

2. **Key Features**:

- Autonomous decision-making capabilities
- Tool integration for external services
- Context maintenance across sessions
- Learning from past interactions

3. **How to Use It**:

- Create specialized agents for different domains
- Implement agent coordination for complex workflows
- Set up monitoring and logging for agent actions
- Define clear boundaries for agent autonomy

4. **Implementation Strategy**:

- Start with a single agent (e.g., `ListingManagementAgent`)
- Define agent goals and capabilities
- Implement tool integration with existing services
- Test agent performance before expanding

Brainstorm how we might use the Responses API and Agents SDK for current and future hub functionalities

Current Hub Functionalities & Proposed Integration

1. **Listing Management**:

- **Current**: Manual relisting via `RelistingAgent`, scheduled jobs through Hangfire
- **With Responses API**: Structured analysis of listing performance
- **With Agents SDK**: Autonomous decision-making for when to relist items

2. **Inventory Management**:

- **Current**: Manual SKU addition, basic inventory tracking
- **With Responses API**: Structured analysis of inventory levels
- **With Agents SDK**: Predictive inventory management based on sales trends

3. **Content Generation**:

- **Current**: Basic title and description generation
- **With Responses API**: More reliable and consistent content generation
- **With Agents SDK**: Context-aware content optimization based on performance

4. **Price Analysis**:

- **Current**: Basic price estimation
- **With Responses API**: Structured market analysis with confidence scores
- **With Agents SDK**: Dynamic pricing based on market conditions

5. **Workflow Orchestration**:

- **Current**: Hangfire jobs for scheduling
- **With Responses API**: Structured workflow decisions
- **With Agents SDK**: Intelligent job scheduling based on priorities

Agent Functionalities

1. **Intelligent Workflow Orchestration**:

- Agents decide which workflows to run and in what order
- Dynamic scheduling based on system state and priorities
- Self-healing workflows that adapt to failures

2. **Market Analysis & Pricing**:

- Continuous monitoring of competitor pricing
- Predictive pricing based on market trends
- Seasonal adjustment recommendations

3. **Inventory Optimization**:

- Predictive reordering based on sales velocity
- Multi-location inventory management
- Slow-moving item identification and action

4. **Customer Service Automation**:

- Automated response to common customer inquiries
- Issue classification and routing
- Sentiment analysis for customer feedback

5. **Performance Analytics**:

- Structured analysis of listing performance
- Identification of optimization opportunities
- A/B testing of different listing strategies

Implementation Roadmap

1. **Phase 1: Responses API Integration** (1-2 months)

- Replace current OpenAI calls with Responses API
- Implement schema-based responses for critical methods
- Add better error handling and retry logic

2. **Phase 2: Agent Implementation** (2-3 months)

- Create specialized agents for each domain
- Implement agent coordination
- Set up monitoring and logging

3. **Phase 3: Workflow Enhancement** (1-2 months)

- Replace manual Hangfire jobs with agent-driven workflows
- Implement intelligent scheduling
- Add self-healing capabilities

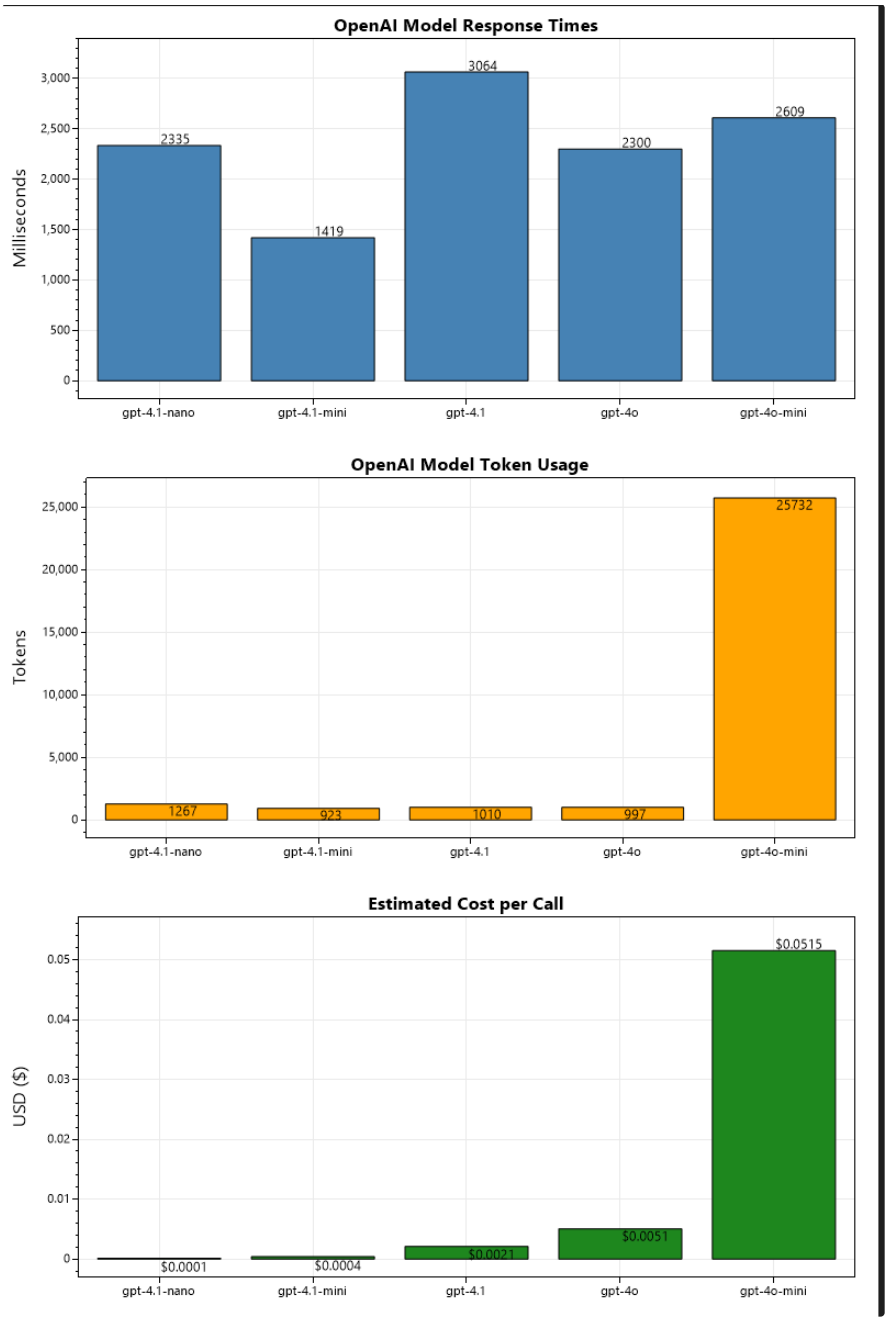
4. **Phase 4: Advanced Features** (3-4 months)

- Implement predictive analytics
- Add customer service automation
- Develop advanced inventory optimization

Performance Benchmark: Latency, Token Usage, and Cost per OpenAI Model"

Cost for each model:

Model	Input ($/1M)	Output ($/1M)
gpt-4.1	$2.00	$8.00
gpt-4.1-mini	$0.40	$1.60
gpt-4.1-nano	$0.10	$0.40
gpt-4o	$5.00	$15.00



What the graph shows

1. Latency (Response Time)

- **Fastest**: `gpt-4.1-mini` (~1419 ms)
- **Slowest**: `gpt-4.1` (~3064 ms)

2. Token Usage

- **Most Efficient**: `gpt-4.1-mini` and `gpt-4o` (~923–997 tokens)
- **Heavy Usage**: `gpt-4o-mini` (~25,732 tokens 😱)

3. Cost

- **Cheapest**: `gpt-4.1-nano` ($0.0001) and `gpt-4.1-mini` ($0.0004)
- **Most Expensive**: `gpt-4o-mini` ($0.0515 😵)

🧠 Combine with Official GPT-4.1 Docs

The GPT-4.1 announcement highlights the following:

Model	Strengths	Ideal Use Cases
GPT-4.1	Top-tier reasoning, 1M context, strong long-context accuracy	Complex document analysis, agents
GPT-4.1-mini	Fast, low cost, better instruction following than GPT-4o	Lightweight apps, chatbots, basic reasoning
GPT-4.1-nano	Cheapest + fast, great for classification/autocomplete tasks	Micro tasks, bulk classification
GPT-4o	Good all-rounder, available in ChatGPT	Most general use cases
GPT-4o-mini	Performs well but high token usage depending on input	🧨 Expensive unless optimized carefully

---

🥇 Best Model Overall (Based on Your Graph + Docs)

✅ **Winner:** `gpt-4.1-mini`

- **Lowest latency**
- **Very low cost ($0.0004)**
- **Moderate token usage**
- According to OpenAI, it *beats GPT-4o in many benchmarks*, including **instruction following** and **format reliability**

> Use this model for: **chatbots**, **internal tools**, **moderate AI agents**, and most real-world scenarios where **response time + cost efficiency** matter.

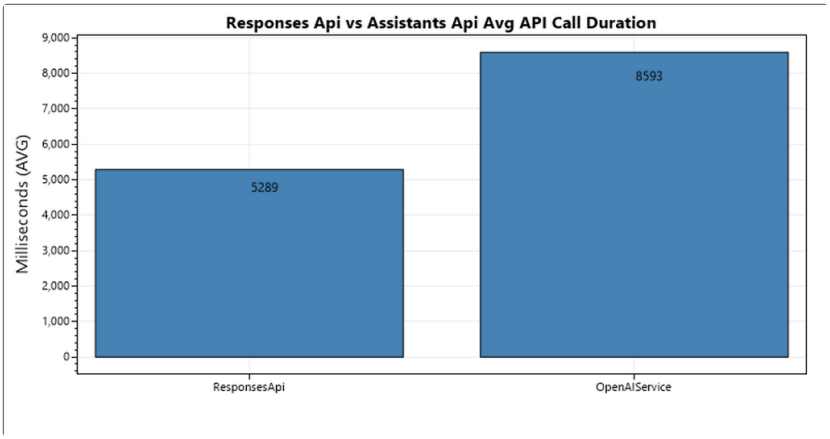
✅ Recommendation Summary

Model	Use It For	Why?
gpt-4.1-mini	General AI agents, chat, apps	Fast, cheap, high-quality instruction following
gpt-4.1	Complex tasks, long context, legal documents	Powerful but slower
gpt-4.1-nano	Classification, autocomplete, minimal tasks	Cheapest and good for micro-queries
gpt-4o	Versatile fallback	Decent on everything, not the best at anything
gpt-4o-mini	❌ Avoid unless optimized	Token explosion + cost overrun

---

Create a test that compares the speed and output of the old functions to the new ones

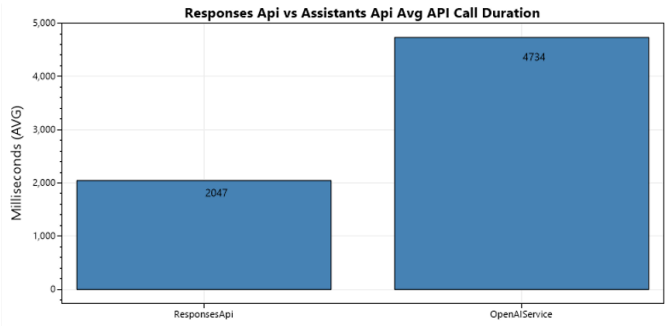
Benchmark: ResponsesApiService vs. OpenAIService (GPT-4o) for Title Generation




```
--- ResponsesApi ---
Avg Time (ms): 5289
Full Output:
<p>This is a versatile HP Photosmart 130 photo printer set, perfect for home printing needs.</p>
<p>Includes HP Photosmart 130 printer, compatible ink cartridge, photo paper, and essential cables.</p>
<p>Comes with a convenient Sunriver Oregon carrying case.</p>
<p>Good pre-owned condition, ensuring reliable performance.</p>
<p>Ideal for creating vivid photo prints effortlessly.</p>
<p>Don't miss out on enhancing your photo printing experience!</p>

--- OpenAIService ---
Avg Time (ms): 8593
Full Output:
<p>This hp photosmart 130 printer is perfect for printing high-quality photos at home.</p>
<p>It comes with a carrying case, power adapter, and additional supplies.</p>
<p>Ideal for photographers and those who love to print memories, this item is pre-owned and
in good working condition.</p>
<p>Don't miss out on this great deal for a reliable photo printer!</p>
```

Benchmark: ResponsesApiService vs. OpenAIService (GPT-4o) for Title Generation



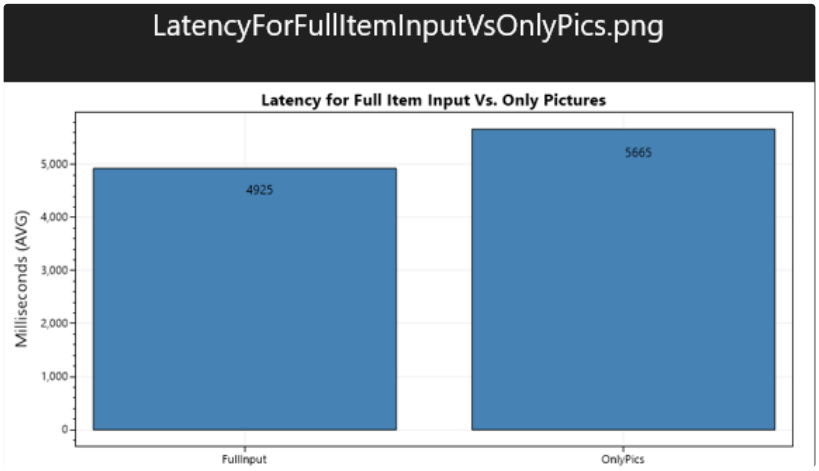
Title_OutputResponses - Notepad

File Edit Format View Help

```
--- ResponsesApi ---
Avg Time (ms): 2047
Full Output:
HP PhotoSmart 130 Compact Photo Printer Bundle - Good Condition

--- OpenAIService ---
Avg Time (ms): 4734
Full Output:
HP Photosmart 130 Photo Printer with Accessories - Great Condition!
```

Benchmark Full Item Input vs Only Pictures as Input



LatencyForFullItemInputVsOnlyPics_Output - Notepad

File Edit Format View Help

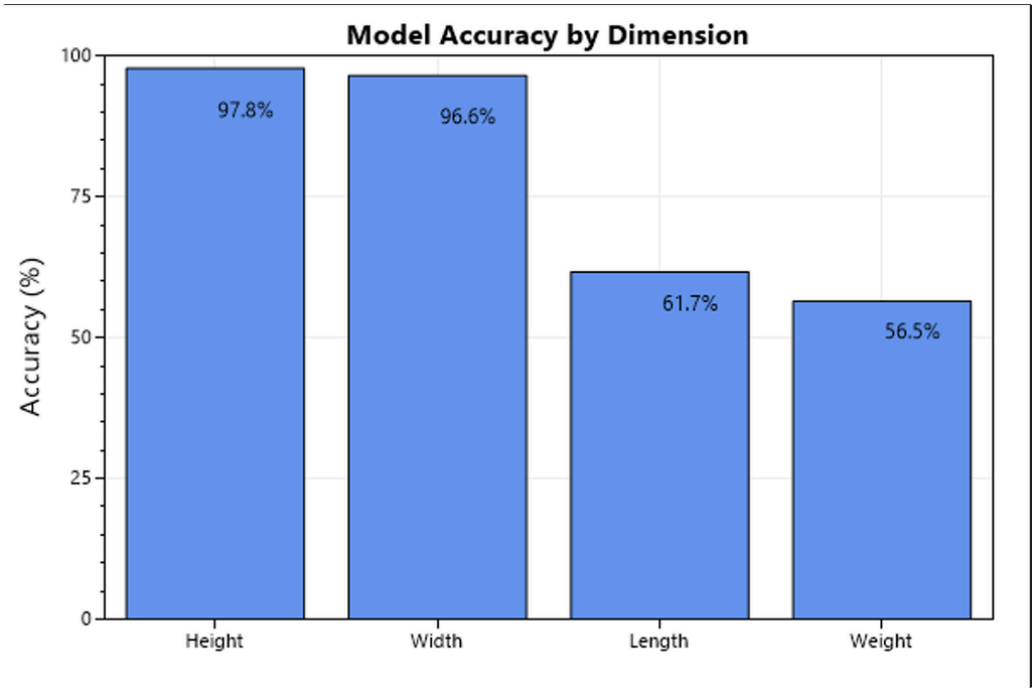
```
--- FullInput ---
Avg Time (ms): 4925
Full Output:
<p>This is a good pre-owned HP photo printer 130 with accessories included.</p>
<p>Includes photo printing guide, power adapter, and photo paper for easy setup.</p>
<p>Comes with an HP 57 ink cartridge for vibrant color printing.</p>
<p>Also includes a Sunriver Oregon carrying case for convenient transport and storage.</p>
<p>Perfect for personal or home office use to print high-quality photos.</p>

--- OnlyPics ---
Avg Time (ms): 5665
Full Output:
<p>This is a pre-owned hp photosmart 135 digital camera printer bundle with accessories.</p>
<p>Includes the hp photosmart 135 printer, hp 57 tri-color ink cartridge, and hp everyday glossy photo paper 100 sheets.</p>
<p>Comes with a Sunriver Oregon branded carrying case for easy transport and storage.</p>
<p>Also included are power cords and a photo printing guide for user reference.</p>
<p>The printer features simple controls and a small LCD screen on the front panel.</p>
<p>Perfect for printing photos directly from a digital camera without needing a computer.</p>
```

Performance Benchmark: GPT-4o-mini Dimension Prediction Accuracy"

We evaluated GPT-4o-mini on 3 items selected from a random slice of 100 records in the Swedemom sandbox database. These items were chosen from the middle of the dataset to avoid duplication and provide diverse test cases. Though the sample size is small (to keep execution time manageable), it demonstrates the model's ability to infer dimensions and weight based on textual and visual input.

In future work, we plan to run broader benchmarks across item categories and explore training a specialized local model. Since we already store verified item dimensions in our dataset, this opens the door to a self-supervised fine-tuning loop or more targeted validation.




```
"reasoning":"The item is a standard DVD case for a movie, typically
measuring about 7.5 inches in height, 5.3 inches in width,
and approximately 0.5 inches in thickness.
This size is consistent for most DVD cases.
Weight is estimated around 0.15 pounds, as DVDs are generally very
lightweight, though exact weight can vary slightly with packaging
and number of discs. Confidence is high for dimensions due to
standardization, and slightly lower for weight due to slight variations."}

{"reasoning":"The item is a special collector's edition set of 6 VHS
tape volumes, each housed in typical VHS-sized cases. A standard VHS
case is approximately 7.5 inches tall, 4.2 inches wide, and about 1
inch thick. Since the 6 volumes are packed in a single box, the box
height matches the VHS height (~7.5 inches), and the width would be
close to the VHS width (~7.5 inches) because the tapes appear to be
placed in two rows or the box is slightly wider than one tape width.
The box depth is estimated as roughly 2.5 inches, enough to hold the six
tapes side by side. The weight is estimated to be around 2 pounds
considering each VHS tape weighs about 0.3 to 0.4 pounds and
packaging adds some weight as well. The confidence for height
and width is high due to direct comparison with known VHS case
dimensions via the images. Depth and weight confidence are slightly
lower as exact packaging and materials are not fully visible."}

{"reasoning":"The item is a DVD set containing 2 DVDs. Standard DVD cases
typically measure approximately 7.5 inches in height, 5.3 inches in width,
and about 0.5 inch in thickness. Since this is a standard DVD case shown
in the images and the description, the measurements align with standard
DVD case dimensions. The weight is estimated based on typical weight
of a single DVD case with 2 discs inside, approximately 0.25 pounds.
The estimation is quite certain for dimensions because these are
standard sizes, while the weight has slightly lower confidence as
actual weight can vary slightly depending on packaging materials."}
```


✅ Moving Forward:
Step-by-Step Path to Automation for Shipping Cost Prediction

---

1. Stick with the Responses API (for now)

You're already getting structured predictions (height, width, length, weight) — this is the hard part and 90% of what you need.

What you're **missing** is:

- Accuracy feedback loop
- Integration with your **shipping estimator**
- Escalation/review logic for bad/confident/unknown cases

---

2. Build a Decision Agent / Pipeline Layer on Top

Use an **agent or orchestration layer** (could just be C# code at first) that does this:

🔁 **For each item:**

- Call the **Responses API** with your schema
- Compare prediction against known examples if available
- If confidence (or similarity to prior items) is high → **auto-approve**
- If not → **queue for manual check or override**
- Pass the estimated dimensions to your **shipping estimator** function (e.g., USPS/UPS/EasyPost)

✅ **You now have an automated shipping cost pipeline with a fallback for edge cases.**

---

3. Improve Over Time via Embedded Retrieval

> "Has this type of item been seen before?"

Add a **vector search layer**:

- Store embeddings of item title + description + category
- Search for **similar past items with known dimensions**
- If found → feed those into your Responses API prompt as guidance

This gives the model better predictions **without fine-tuning**.

---

4. Optional: Use an Agent Later for Multi-Step Reasoning

If you want the model to:

- Search past items
- Decide whether to predict vs retrieve
- Estimate shipping cost directly

... then yes, you can build an **OpenAI Assistant (agent)** with:

- A **vector search tool**
- Your existing **shipping cost estimation function** (as a tool)
- Instruction like:

  "Given an item title and description, predict dimensions using prior examples, then estimate shipping cost using our API. If confidence is low, escalate."

But this is a **Phase 2 or Phase 3 enhancement**, not needed to go live.

Item → Responses API (structured dimensions)
→ If similar item found → reinforce or override
→ Estimate shipping (EasyPost, etc.)
→ Auto-list or escalate for manual check

🧠Case Study: How does Amazon estimate Dimensions/Shipping Costs

Amazon has built internal models and published whitepapers, including:

📄 Whitepaper: Rate Card Transformer (RCT)

- Uses **self-attention** (like in GPTs) to estimate shipping cost from:
  - Product metadata
  - Carrier rate cards
  - Historical fulfillment data
- Achieves **~29% lower prediction error** than tree models

📄 How Amazon Uses AI to Reduce Packaging

- Describes internal tooling to:
  - Predict optimal packaging
  - Reduce overuse of materials
  - Identify dimensional weight impact

> ❗ However, **there's no whitepaper on predicting item dimensions from scratch**, likely because Amazon **relies heavily on image-based estimation** in warehouses — something they haven't open-sourced.

🔍 So What Did Amazon Do Before AI Estimation?

They:

- Required sellers to enter dimensions manually
- Fell back to fulfillment center measurement (slow + costly)
- Used standardized sizes for common barcoded goods
- Heavily penalized incorrect inputs to encourage accuracy

🧩 So What Can You Do (Mirroring Amazon's Internal Evolution)?

You can do the same in leaner form:

Amazon's Tool	Your Equivalent
Barcode lookup	SQL or vector search of past items
FBA measurement	Human override queue if confidence is low
AI packaging engine	GPT-4o + structured schema + photos
Rate Card Transformer	Call EasyPost or USPS API with dims
Packaging optimization	Store shipping class (first class, parcel, etc.) and revise over time

---

✅ Recommendation for You

There are **no public whitepapers specifically on dimension estimation via LLMs** (yet), so you're pioneering this in a powerful way:

- Keep your **Responses API setup**
- Layer in a **confidence-based approval loop**
- Optionally build a **training set** to experiment with local fine-tuning or vision models later (e.g., [YOLOv8 + object detection + GPT prompt generation]).