ChatGPT

Key Changes from GPT-3.5/4 to GPT-5 in OpenAI's API

1. API Endpoint Evolution: Chat Completions to Responses API

Chat Completions vs Responses: Earlier GPT-3.5 and GPT-4 (including GPT-4-Turbo) use the Chat Completions API (`/v1/chat/completions`) for conversational interactions. GPT-5 introduces a new **Responses API** (launched in 2025) that unifies chat and tool usage features [1] [2]. GPT-5 *can* still be called via the Chat Completions endpoint for backward compatibility, but OpenAI **recommends using the Responses API** to unlock its full capabilities [3] [4]. The Responses API is stateful – each response has an `id`, and you pass a `previous_response_id` on the next call to maintain context, rather than resending the entire conversation each time [5]. This is a shift from chat completions, where the developer had to include the full message history on every request. In short, migrating to GPT-5 often means switching to the new Responses endpoint to benefit from built-in state management and advanced tool integration.

New Endpoints: The Responses API (accessible via OpenAI's SDK as `client.responses.create`) combines features of the old chat and the beta Assistants API. It returns richer structured outputs and supports extended functionality (see below). GPT-5 is available on both endpoints (Responses and Chat) in three sizes (`gpt-5`, `gpt-5-mini`, `gpt-5-nano`) [3], but using Responses yields lower latency and better tool execution performance [6]. In summary, **the main endpoint change** is the introduction of the Responses API – tools and agents built on GPT-3.5/4's chat API should be updated to use this new endpoint (or at least accommodate its output format) when upgrading to GPT-5.

2. Parameter Changes and Deprecations in GPT-5

GPT-5 brings **breaking changes to several API parameters**. Notably, the parameter names and supported options have changed in ways that will break older integration code if not updated:

- **Max Tokens:** The familiar `max_tokens` parameter is **no longer supported** for GPT-5 calls [7]. Attempts to use it will result in an error (`"Unsupported parameter: 'max_tokens' is not supported with this model. Use 'max_completion_tokens' instead."` [8]). The replacement is `max_completion_tokens`, which serves the same purpose (limiting the length of the model's output) but must be used for GPT-5 models [9]. This change was introduced in the GPT-5 and related "o-series" models, so any code using `max_tokens` needs to switch to `max_completion_tokens` when targeting GPT-5.

- **Temperature and Top-p:** GPT-5 **does not support** the `temperature` and `top_p` parameters (used for controlling randomness and sampling) [7]. These stochasticity controls have been removed for the GPT-5 family – passing them yields errors or is simply ignored. In practice, this means GPT-5's generation style is managed through other means (like the new *reasoning and verbosity* controls, see below) rather than raw randomness knobs. Developers migrating from

GPT-3.5/4 should drop any use of `temperature`, `top_p` (and by extension, the related `frequency_penalty` and `presence_penalty` which are also unsupported) [10]. GPT-5 outputs tend to be more deterministic and consistent, and you cannot directly make it "more creative" or "more conservative" via temperature – you'd instead use the model's new parameters or prompt techniques to influence style.

- **New Parameters – Verbosity & Reasoning Effort:** GPT-5 introduces new parameters to control output length and cognitive effort. The `verbosity` parameter (values: `"low"`, `"medium"`, `"high"`) hints how expansive or concise the answer should be [11] [12]. For example, `verbosity="low"` yields terse answers, while `"high"` produces very detailed, verbose explanations [13] [14]. This replaces the need to manually engineer prompt length or rely on token limits to shape response length – developers can now explicitly request short vs. long answers. The `reasoning_effort` parameter controls how much reasoning or step-by-step thought the model invests before answering. GPT-5 adds a new `"minimal"` level (in addition to `"low"`, `"medium"`, `"high"`) [15]. Using `reasoning_effort="minimal"` makes GPT-5 skip or shorten its internal reasoning process to respond faster with minimal analysis [16]. This is ideal for simple tasks or when latency is critical, but it's not recommended for complex, multi-step problems [16]. By default GPT-5 uses medium reasoning effort. (In effect, high reasoning effort is akin to GPT-4's thorough chain-of-thought, whereas minimal makes it behave more like a quick heuristic answerer.)

- **Model Support Matrix:** It's worth noting that the smaller GPT-5 variants and transitional "GPT-4o" models have similar parameter restrictions. For example, the OpenAI "o-series" reasoning models (often referred to as GPT-4o, like `o1`, `o3`, `o4`) also dropped `max_tokens` in favor of `max_completion_tokens` and do **not** support temperature/top-p [7] [17]. They partly introduced the `reasoning_effort` concept prior to GPT-5. When upgrading, assume **GPT-5 and its related new models require these new parameter conventions**. Any code passing deprecated params (max_tokens, temperature, top_p, presence_penalty, etc.) must be updated or removed. Instead, embrace `max_completion_tokens` for length, and use `verbosity` for output length/style and `reasoning_effort` for speed-vs-quality tuning.

Summary of Parameter Updates: To upgrade to GPT-5, ensure your API calls use the new parameter names and defaults: - Replace `max_tokens` with `max_completion_tokens` [18]. - Remove `temperature`, `top_p`, and similar parameters (GPT-5 won't accept them) [18]. - Incorporate `verbosity` (optional) to control answer detail [11]. - Incorporate `reasoning_effort` (optional) – e.g. use `"minimal"` for quick, low-latency replies [19]. These changes are **not backward-compatible** with older models, so you may need conditional logic if supporting both GPT-4 and GPT-5 in the same codebase.

3. Tool Calling and Function Call Handling Changes

One of the biggest shifts from GPT-3.5/4 to GPT-5 is how the API handles function/tool calls:

- **Parallel and Chained Tool Calls:** Previous models could call at most one function at a time per turn (GPT-4's function calling would return a single `function_call` payload, after which the developer executed it and then called the model again). GPT-5, especially via the Responses API, is far more flexible – it can **issue multiple tool calls in one response and even call tools in parallel** [20] [2]. The model's improved "tool intelligence" means it can plan a sequence of actions and either execute

them step-by-step or sometimes concurrently (if supported). For example, GPT-5 might decide to run a web search and a database query in parallel to save time, whereas GPT-4 would have done them sequentially. There is a new boolean parameter `parallel_tool_calls` to enable/disable the model's ability to call tools simultaneously [21]. By default, parallel calling is allowed; if your application isn't ready to handle concurrent tool results, you can disable it. Upgrading an agent to GPT-5 requires checking for **multiple tool invocations** coming back and handling them appropriately (e.g. spawning parallel execution threads or queueing them as needed).

- **"Custom" Tools & Free-Form Function Calls:** GPT-4's function calling required every tool's input and output to be structured as JSON per a defined schema. This sometimes made code generation or command-line tool use clunky, because the model had to fit everything into JSON strings. GPT-5 introduces **free-form function calling** for custom tools: you can register a tool with `type: "custom"` that allows the model to output an arbitrary plaintext payload (like a chunk of code, SQL query, or shell command) instead of a JSON object [22]. This means GPT-5 can write code or other text directly as a tool input without JSON-escaping issues, which *greatly* simplifies coding agents [23]. For instance, instead of returning `{"script": "<PYTHON CODE>"}` as a JSON argument for a code execution tool, GPT-5 can directly produce the code string for a custom `run_python` tool. If you have existing function definitions from GPT-3.5/4, they remain compatible (standard JSON-based functions still work in GPT-5) [24], but you now have the **option** to define custom tools when unstructured input is more natural. Migrating an agent to GPT-5 might involve converting some tools to `type:"custom"` (e.g., for a shell command tool or a free-form text generator) so the AI can operate more freely.

- **Built-in Tools and Tool Types:** OpenAI has expanded the set of **built-in tool types** available to GPT-5. The Responses API natively supports certain tools like `code_interpreter` (for running Python code in a sandbox), `web_search`, `file_search`, `image_generation`, etc., without the developer having to implement them [2]. GPT-5 is trained to use these tools effectively and can chain them as needed [25]. In contrast, GPT-3.5/4 had no built-in tools (aside from the limited browsing plugin in ChatGPT or developer-provided functions) – all functionality had to come from user-defined functions or external plugin systems. With GPT-5, your agent can leverage OpenAI-provided tools out-of-the-box. For example, GPT-5 might invoke a `file_search` tool to find relevant code files, then a `code_interpreter` to run tests, all within one session [2]. To use these, you must enable them in the request (typically by including them in the `tools` list when calling the API). **Recommendation:** When upgrading, consider using built-in tools for common tasks (search, code exec, etc.) instead of custom implementations – this can reduce development overhead and take advantage of GPT-5's training. Also be prepared for GPT-5 to return tool call results of types you may not have seen before (e.g., it might return an `image_generation` call if the user asks for an image, whereas older models would just say they cannot do that).

- **Tool Call Return Format:** The structure of how a tool/function call is returned has changed. In GPT-4's Chat Completions, a function call appeared as part of the assistant's message with a `function_call` field (containing the function name and arguments JSON). In GPT-5's Responses API, **tool calls are returned as separate items in the response output list** (see next section on response format). Each tool call item includes fields like `type` (identifying it as a function call, code interpreter call, etc.), a unique `call_id` to track it, the tool `name`, and the `input` or `arguments` for the tool [26] [27]. After executing the tool, the developer sends the result back to GPT-5 in a follow-up call, similarly identified by that `call_id` [28] – but instead of crafting a chat

message with role “function”, you typically send a `type: "function_call_output"` item containing the output and the `call_id` [29]. **In summary**, GPT-5’s tool calling is more complex (multiple and parallel calls possible, with richer types) but also more powerful. Upgrading agents means handling these multi-step interactions: parse possibly multiple tool calls from one response, execute them, and return results in the new expected format. Also account for *preamble messages* (explained next) that the model might output about its plan before actually calling a tool.

4. Response Schema and Format Differences

Structured Output List: Unlike the older chat API which returned a single message (or a list of choice messages) per request, the GPT-5 Responses API returns a **structured list of outputs** in each response. The `response.output` field is an ordered list where each element can be one of several types (e.g. a user-visible assistant message, a tool/function call, a reasoning summary, etc.) [30] [31]. For example, GPT-5’s answer might consist of an *assistant message* explaining what it’s doing, followed by a *function_call* item for a tool, then another message with final results. This is a significant change from GPT-3.5/4 where the API response would typically contain just one `assistant` message (with an optional function_call).

Each item in `response.output` has a `type` **field** designating its role: - `type: "message"` – a normal message from the assistant that should be shown to the user (contains text content and a role, usually `"assistant"`) [32] [33]. - `type: "function_call"` (or other tool types like `"function_call"`, `"code_interpreter"`, `"image_generation"`, etc.) – indicates the model is calling a tool, with fields such as `name`, `arguments` (for JSON calls) or raw `input` (for custom calls), and a `call_id` [27]. These are actionable items for the application rather than end-user content. - `type: "reasoning"` or `"reasoning_summary"` – these are internal reasoning traces or preamble explanations. GPT-5 can output *reasoning summaries* or plan descriptions as special items (often not meant as final user output but for transparency or debugging) [34] [31]. For instance, a “reasoning” item might contain the model’s summary of its plan before it proceeds to tool use.

Additionally, each item may carry a `status` (e.g. `"completed"` when a message or function call segment is finished) [32] [27]. The response also includes an `id` for the whole response and usage metrics (token counts) similar to before.

Impact on Parsing: Developers must adapt any code that parses model outputs. In the old API, one would typically take `response.choices[0].message.content` as the assistant’s answer. In the new format, you need to iterate through `response.output` and handle each element: - Concatenate or display all **assistant message** portions in order (GPT-5 might split a long answer into multiple message chunks or intermix them with tool calls). - Detect **tool call** items and trigger the appropriate tool actions. (Each call has a unique `call_id` to correlate with its eventual result.) - Possibly handle or skip **reasoning/preamble** items. These can be shown to the user as the model’s “thinking out loud” (for transparency) or logged for debugging. GPT-5 is designed to sometimes output a brief plan summary for the user *before* using tools [34] [31] – something prior models did not do. Your interface might choose to present these to users (to improve understanding of the AI’s process) or ignore them in favor of only final answers.

No More `finish_reason` **in Choices:** In Chat Completions, each choice had a `finish_reason` (e.g. “stop” or “function_call”) to indicate why it ended. In the Responses API, the concept is a bit different. The end of a response is implicit when the `output` list is fully processed and no further tool calls are pending.

The model can explicitly signal it's done when it has no more actions – for example, by returning a final `"message"` with a `"status": "completed"` and no subsequent items. When migrating, you'll remove logic that checks `finish_reason`; instead, you loop until the model gives no new output items or until you decide to stop (the new API often involves a loop of: send user input → get outputs including tool calls → execute tools → call model again with results → repeat, until no more tool calls).

Stateful Conversations: As noted, the Responses API keeps track of conversation state server-side using `previous_response_id`. This means the **conversation history does not need to be passed back and forth in full**. In GPT-3.5/4, you accumulated a list of messages (system, user, assistant, etc.) and sent that with each request. With GPT-5, you typically send only the new user message (and any tool results as needed) along with the last response's ID, and the API carries over the context (including the model's prior reasoning, which isn't fully visible to you but is preserved) [35] [36]. This greatly reduces prompt size and token usage over long sessions [37]. **Migration consideration:** If your app currently maintains a `messages` array, you can simplify: just store the last response ID and the conversation content you need for your own display, and use `previous_response_id` to continue the convo. If you continue using the Chat Completions endpoint with GPT-5 (not recommended, but possible), you won't have this feature – you'd still send messages each time, but be aware that GPT-5's response might contain function call data in a slightly different shape even on the chat endpoint.

Streaming Mode: Streaming responses is still supported with GPT-5. You initiate streaming the same way (e.g. `stream=True` in the API call) and receive incremental tokens. However, with the new output format, streaming may deliver a sequence of *items*. In practice, GPT-5 will stream out the content of a message item token-by-token just like before, then indicate the end of that item and potentially begin streaming a tool call item, etc. The OpenAI SDK abstracts some of this, but if you have a custom streaming parser, you should be prepared for the stream to yield structured data (e.g., you might get a chunk indicating a function call start). The general approach to updating streaming code is similar to non-stream parsing: handle different item types as they arrive. If your older tool simply streamed the assistant's message text, now you may need to filter out tool-call JSON from the stream or present interim reasoning. In essence, **streaming with GPT-5 remains possible and recommended for latency, but the application must handle multi-part outputs** (you might, for example, stream out the assistant's plan message to the user, then pause while a function is being executed, then continue streaming the next part of the answer).

5. GPT-5 Model-Specific Features and Behaviors

GPT-5 isn't just a drop-in replacement – it has new capabilities and some differences in behavior that developers should note:

- **Expanded Context Window:** GPT-5 offers a much larger context window than GPT-4. While GPT-4 topped out at 8K tokens (and 32K in a special version), GPT-5 supports dramatically larger contexts (on the order of 100K+ tokens) [38] [39]. In practical terms, this means GPT-5 can ingest and produce much longer documents or hold larger conversations without losing context. Tools and agents migrating to GPT-5 can take advantage of this by allowing larger inputs (e.g., feeding whole files or extensive logs to the model). However, be mindful of cost – more context means more tokens billed. Also ensure any hard-coded limits or chunking logic in your app (designed for a 4K/8K limit) are revisited, as they might be unnecessarily strict when using GPT-5.

- **System Message and Instruction Following:** The role of the system message remains important in GPT-5 – it's how developers impose high-level behavior or persona. GPT-5 has been trained to follow system instructions even more robustly and also to **communicate its actions clearly** [12] [20]. There aren't breaking changes to how you provide system prompts (you still include a system role message at the start of the conversation input for the first turn). However, because the conversation can be continued via `previous_response_id`, you typically set the system message once at the beginning and do not need to resend it on every turn (unlike in stateless chat completions). GPT-5 also introduces the concept of **tool preamble messages**: these are *assistant-originated* messages that precede tool use, explaining what the AI is about to do. They are not exactly system messages, but they are a new type of model behavior where GPT-5 will proactively outline its plan ("I'm going to do X, then Y…") to the user [34] [31]. This transparency was not present in GPT-3.5/4, which tended to either just do the tool call or produce hidden reasoning only. Now the system developer can encourage or configure the style of these preambles via the prompt (e.g. provide guidelines in a `<tool_preambles>` section of the prompt) [40]. In summary, system messages work similarly, but you may need to adjust them to account for GPT-5's extended abilities (for instance, instructing how verbose to be in explanations, since GPT-5 can explain steps to the user).

- **Reasoning and Autonomy:** By design, GPT-5 is **more "agentic"** – it can handle long, complex tasks with many steps without human intervention. It's better at sticking to a goal through multiple tool calls and fewer clarification questions [20]. In practice, if you set `reasoning_effort` high or use the default medium, GPT-5 will try harder to solve a problem fully on its own. This means an agent based on GPT-5 might **ask the user fewer follow-up questions** or confirmations compared to GPT-4. If your tool was relying on the model to defer to the user frequently, be aware GPT-5 may instead push forward with assumptions (this can be tuned via prompts or by using minimal reasoning mode if you want more lightweight behavior) [41] [42]. GPT-5's greater autonomy is generally a feature (it completes tasks end-to-end more often), but ensure your application's safety or business logic can handle this (for example, set boundaries on actions if needed, since the model might attempt more powerful tool uses in pursuit of a goal).

- **Multimodal and Tool Modalities:** GPT-5 in the API is primarily a text model, but it's mentioned as handling "text, image, audio, and video" inputs/outputs via tools [43]. It can employ the `image_generation` tool to create images, and an `audio_transcription` tool (or similar) to process speech, etc. While GPT-4 had a vision-enabled version (GPT-4V) and some audio capabilities (Whisper API) separate from the chat model, GPT-5 consolidates multimodal abilities. If your existing application was text-only, you might consider whether GPT-5's users can now ask for other modalities (e.g. "show me a diagram" – GPT-5 might try to call the image generator). Plan for how to handle such outputs (display images if your UI can, or politely refuse if not supported). Also, GPT-5's improved *factuality* and reduced hallucination rate [12] [44] means it's more reliable, but it's still good practice to keep any verification steps you had for GPT-4 if critical.

- **Streaming and Background Tasks:** GPT-5 supports a new "background mode" where long-running operations can be handled asynchronously [45]. This is more an enterprise feature, but essentially the model can offload lengthy tool sequences and return immediately with a job ID, then later provide the result. Most simple agent implementations might not use this, but if your GPT-4-based tool had to, say, poll for long outputs or manually chunk work, GPT-5 offers API-level help. There's also support for **reasoning summaries** – GPT-5 can return an encrypted or structured summary of its reasoning process for audit without revealing full chain-of-thought, which might be useful in

regulated contexts [45]. These aren't breaking changes per se, but new features to be aware of when upgrading (you likely won't encounter them unless you opt-in via parameters).

- **Token Usage Differences:** With the removal of sampling parameters and introduction of reasoning control, how GPT-5 uses tokens may differ. In minimal reasoning mode, it produces far fewer "thinking" tokens (those invisible chain-of-thought reflections) and jumps to answers more quickly [16], which can save token costs on simple tasks. On complex tasks with high reasoning, GPT-5 might consume more tokens planning and explaining than GPT-4 would, but often it compensates by avoiding redundant back-and-forth or mistakes. Also, thanks to `previous_response_id` statefulness, you avoid resending the full history, cutting down prompt size significantly in multi-turn scenarios [46] [37]. Overall, after upgrading you might observe **different token usage patterns** – it's wise to re-evaluate any token limits or budgeting logic (for example, if you had logic to truncate history to stay under 4096 tokens, you may not need that as often with stateful context and larger windows).

6. Transition & Compatibility Guidelines for Upgrading to GPT-5

Upgrading a tool or agent from GPT-3.5/GPT-4 to GPT-5 will require careful adjustments. Below is a summary checklist of **breaking changes to address and best practices** to follow:

- **Switch to the Responses API (if possible):** To fully leverage GPT-5, migrate your API calls from `chat.completions.create` to `responses.create` in the OpenAI SDK [47] [48]. This involves using the new request format (`model`, `input`, `tools`, etc.) and handling `previous_response_id`. If you continue with the Chat Completions endpoint, be aware you'll miss out on stateful context and some tool features, and you must still conform to new parameter requirements (no `max_tokens` / `temperature`). Using the Responses API is recommended for performance and capability reasons [6] [4].

- **Update Parameter Usage:** Refactor your API request construction to use the new parameters. Remove any references to `temperature`, `top_p`, `presence_penalty`, etc., as these will cause errors with GPT-5 [7]. Replace `max_tokens` with `max_completion_tokens` everywhere you set it [18]. Start incorporating `verbosity` and `reasoning_effort` settings based on your use case:

- For instance, if your old app sometimes truncated outputs to force brevity, you can now simply set `verbosity: "low"` in the request [11].

- If you had a "fast mode" vs "accurate mode", consider mapping that to `reasoning_effort: "minimal"` vs `"high"`, respectively. Expose these controls to users if appropriate (the reference implementation for GPT-5 CLI tools adds command-line flags for them [49]).

- **Adapt to New Response Structure:** Rewrite any logic that parsed the model's reply. Instead of assuming one `assistant` message content, loop through `response.output`. For each element:

- If it's a `"message"` with role `assistant`, display or store its `content.text` as part of the assistant's answer.

- If it's a `"function_call"` (or other tool call), dispatch it to your tool executor. Use the provided `name` to identify which tool to run, and parse the `arguments` (which might be JSON string) or raw `input` (for custom tools). Note the `call_id` and expect to include that in your follow-up when returning results [26] [28].
- Do **not** assume only one function call per turn – handle multiple. Possibly execute them sequentially unless you've built concurrency (GPT-5 might return two calls intended to run in parallel to save time).
- If an item is of type `"reasoning"` or `"summary"`, you can log it or present it (these are the model's explanatory notes). They may be marked with a role like `"assistant"` but semantically are out-of-band comments. Decide if your UX should show these to end users or use them for debugging. At minimum, don't confuse them with final answers.

- Continue the loop of model → tool → model until you receive a response that has no further tool calls. GPT-5 will finish the exchange by outputting a final assistant message with no pending calls (the PDF mentions a "finish signal" when no tool calls remain) [50]. Your agent should detect that and conclude the cycle.

- **Tool Definitions and Execution:** Your existing tool (function) definitions can mostly carry over. The JSON schema format for function tools remains the same, and you'll still provide the list of tools on each call (or register them in the assistant) [24]. However, take advantage of new types:

- Where appropriate, define some tools as `type: "custom"` so the model can output free-form text for them [23]. For example, if you had a "execute shell command" tool that previously took a JSON with a `command` field, you might redefine it as a custom tool so GPT-5 can just produce the shell command string directly. This reduces friction in generation.
- If you need to enforce a strict format for tool inputs or outputs, consider using **Context-Free Grammars (CFGs)**. GPT-5 lets you attach a CFG or regex pattern to constrain what it can output for a tool [51]. This is useful if your agent expects, say, a specific config file syntax or code structure. It wasn't really possible in GPT-3.5/4 to *guarantee* format beyond basic stop tokens, so this is a new capability.
- **Built-in vs Custom:** Evaluate if any of your custom tools overlap with the new built-in ones. For example, if you built a web search integration for GPT-4, you might switch to the official `web_search` tool provided by OpenAI (letting the model call it directly) [2]. The same goes for code execution or file retrieval – using OpenAI's tools could improve reliability since the model was trained with them in mind. You might keep some local tools if they do something very specific for your app, but you may register them differently now (the API supports remote tool plugins via MCP – model protocol – if needed) [52] [53].

- Also note GPT-5 can handle **multiple tools returning results at once**. For example, if two parallel calls were made, you might need to send two `function_call_output` messages back (or a combined payload with both results). Ensure your tool execution pipeline can queue or manage simultaneous calls and return the outputs properly tagged with their `call_id` [29].

- **Testing and Validation:** Because of these changes, it's critical to test your upgraded agent thoroughly. Expect to uncover edge cases:

- If your code assumed one response message, see what happens when GPT-5 provides a reasoning preamble plus an answer – are you accidentally showing the user duplicate info or missing the actual answer because you only took the first item? Adjust accordingly.
- If your agent logic wasn't prepared for tool calls without an explicit user prompt (GPT-5 might proactively use tools more often), make sure that doesn't break flow. For instance, GPT-5 might decide to use the `web_search` tool in cases GPT-4 would have just said "I don't have that information" – ensure your app has the necessary API keys or permissions set up if you enable such tools.
- Verify that removing temperature/top_p doesn't adversely affect functionality that relied on those (e.g. variability in storytelling). You might need to compensate with prompt tweaks or by using the verbosity setting (which can indirectly influence style) [14]. In the OpenAI developer community, some noted GPT-5 can be "too factual" or rigid without temperature; if creative output is needed, consider instructing the model via the prompt to be imaginative, since you can't just dial up randomness.

- Check performance and costs: GPT-5 is more expensive (in token pricing) than GPT-4. Monitor that the `reasoning_effort` level you choose provides a good quality/cost trade-off. For quick tasks, use minimal reasoning to save tokens [16]; for critical tasks, use higher reasoning but perhaps limit context or instruct the model not to over-explain to save output tokens.

- **Backward Compatibility:** If you need your tool/agent to support multiple model versions (e.g. some users on GPT-4, others on GPT-5), you may have to maintain dual code paths. For example, one path constructs the old style request and parses accordingly for GPT-4, and another path for GPT-5. The GitHub issue for DSPy library provides a small matrix that highlights differences (GPT-4 supports `temperature` but GPT-5 does not, etc.) [17]. You can use model name checks (e.g., if model name starts with "gpt-5" or matches the new patterns) to toggle which parameters and parsing logic to use [54]. Over time, OpenAI may fully retire older APIs, so it might be simpler to upgrade fully to GPT-5 and drop support for older models if that's feasible for your project.

In conclusion, migrating from GPT-3.5/4 to GPT-5 involves **significant but manageable changes**. The main breaking changes include using the new Responses API (or adapting to its output structure), renaming/removing certain parameters (`max_tokens`, `temperature`, etc.) [18], and handling a more complex tool interaction scheme (parallel calls, custom tool outputs, and new response item types). By updating your API calls, adjusting your parsing of responses, and leveraging GPT-5's new features (verbosity control, reasoning levels, built-in tools), you can make your AI tool both compatible with GPT-5 and more powerful than it was with earlier models. These updates will ensure your application continues to work reliably with GPT-5 and takes full advantage of its capabilities [55] [56].

Sources:

- OpenAI announcement: *"Introducing GPT-5 for Developers"* – highlights new API features (verbosity, minimal reasoning, custom tools) and recommends using the Responses API [12] [3].
- OpenAI Cookbook: *"GPT-5 New Params and Tools"* – outlines the new parameters (`verbosity`, free-form function calling, CFG constraints, `reasoning_effort`) and supported endpoints [57] [58].
- GitHub issue (stanfordnlp/dspy#8612) – documents breaking changes in GPT-5's API (deprecation of `max_tokens`, lack of `temperature/top_p`, introduction of `max_completion_tokens`, etc.) [7].

- *GPT-5 Prompting Guide* (OpenAI Cookbook) – provides examples of GPT-5's response format with reasoning and tool call items, and guidance on using `previous_response_id` for state [59] [36].
- *API analysis and porting plan (Grok CLI → QuietEnable)* – an internal report detailing how to adapt an existing GPT-4-based CLI agent to GPT-5; it enumerates changes to API calls, conversation handling, and tool definitions [1] [60].
- OpenAI community & Azure notes – confirm that GPT-5 and related models require `max_completion_tokens` and drop old parameters, aligning with the above (e.g., Azure OpenAI's `o1` / `o3` models had similar constraints) [61] [62].
- Encord technical blog: *"GPT-5: A Technical Breakdown"* – summarizes GPT-5's improvements (better tool use, up to 128K context) and assures GPT-5 works with prompts made for GPT-4Turbo with improved results [39] [43].

---

[1] [2] [3] [5] [6] [11] [15] [19] [21] [22] [23] [24] [25] [26] [28] [29] [30] [35] [45] [46] [47] [48] [49] [50] [51] [52] [53] [55] [56] [60] API analysis and porting.pdf
file://file-UkAEi67g5nX2qFB89kVxfY

[4] [13] [14] [16] [57] [58] GPT-5 New Params and Tools
https://cookbook.openai.com/examples/gpt-5/gpt-5_new_params_and_tools

[7] [8] [9] [10] [17] [18] [54] [61] [62] [Bug] DSPy incorrectly passes `max_tokens` to OpenAI GPT-5 models instead of `max_completion_tokens` · Issue #8612 · stanfordnlp/dspy · GitHub
https://github.com/stanfordnlp/dspy/issues/8612

[12] [20] [44] Introducing GPT-5 for developers | OpenAI
https://openai.com/index/introducing-gpt-5-for-developers/

[27] [31] [32] [33] [34] [36] [37] [40] [41] [42] [59] GPT-5 prompting guide
https://cookbook.openai.com/examples/gpt-5/gpt-5_prompting_guide

[38] [39] [43] GPT-5: A Technical Breakdown
https://encord.com/blog/gpt-5-a-technical-breakdown/