ChatGPT

Strategy to Win the MCP Tool/Server Hackathon Track

Understanding the Hackathon and Available Credits

The **Gradio "Agents & MCP" Hackathon 2025** offers a dedicated track for building an MCP Tool/Server. In **Track 1 (MCP Tool/Server)**, participants create a Gradio app that also functions as an MCP server – essentially a specialized tool that an AI agent can use [1]. You're competing solo in this track, which is well-suited to your Python and back-end skills (as opposed to Track 2's frontend component building or Track 3's broad demos [2]).

Critically, the hackathon provides *generous free credits* from sponsors to empower your project. Every participant receives **$250 in Modal Labs compute credits** (for cloud CPU/GPU usage) and **$25 in Hugging Face API credits** [3]. Early registrants also get API credits from others: e.g. OpenAI and Anthropic ($25 each to first 1000), Nebius cloud ($25 to first 3300), Mistral AI ($25 to first 500), and SambaNova ($25 to first 250) [4] [5]. These credits are a huge boon – effectively **$300+ of cloud and AI services** at your disposal. The key is to leverage them smartly to build an impressive project.

Finally, note the judging structure: each track has a $2,500 first prize and $500 second prize, plus special sponsor awards for things like best use of Modal, LlamaIndex, etc. [6]. This means using the sponsors' tools effectively (Modal's platform, LlamaIndex for data, etc.) can increase your chances of winning **either the track or a sponsor's special award**.

Choosing a Winning Project Idea (MCP Server for ICD-10 Coding)

Given your background and interests, a strong idea is to build an **MCP server for medical coding**, specifically focusing on **ICD-10 diagnosis codes**. This aligns with your excitement about ICD-10 and can showcase a practical, high-impact use of AI. In healthcare, translating doctors' free-text notes into standardized ICD-10 codes is a time-consuming but critical task for billing and analytics [7] [8]. There are ~68,000 ICD-10-CM codes in use [9], and coding is **non-trivial for humans** – it requires expert training and careful lookup of complex guidelines [8]. An AI-powered tool to assist with this would clearly demonstrate value.

Importantly, **AI for ICD-10 coding is a hot topic**. Research shows automated coding could reduce clinician burden and improve accuracy [10]. In fact, one study found ChatGPT-4 could achieve up to 99% accuracy on certain specialty coding tasks [11] [12], underscoring the potential. Another study, however, found that a *retrieval-based approach* outperformed a generative model on broader coding data: using an embedding model to find the closest ICD-10 code achieved ~80% accuracy, while GPT-4 only reached ~50% on the same task [13]. This suggests a **hybrid approach** (semantic search + AI reasoning) might be most robust. As an MCP tool, your server could combine both: search a database of ICD codes for likely matches, and optionally use an LLM to refine or explain the results.

Crucially, this idea leverages your **Python expertise** and does not require extensive frontend work. You'll build a backend service (with a simple Gradio UI for demo) that an LLM agent can query. It's aligned with Track 1's aim to *"extend the capabilities of your favorite LLM"* via a tool [1] – here, the capability is medical code lookup/assignment. Moreover, it's a fairly unique niche; while some entrants might build general-purpose tools, a focused healthcare tool can stand out. (There are early examples of MCP servers in healthcare – e.g. one provides drug info, PubMed search, and an ICD-10 lookup feature [14] – showing that this domain is recognized as useful for MCP, but your project can differentiate by depth in ICD-10 coding.) Overall, **ICD-10 coding assistance** is impactful, showcases AI strengths, and gives you a clear scope for the one-week hackathon.

Leveraging Free Credits and Technologies Effectively

To maximize your chances, you should **strategically use the free credits** and sponsor tools in building this project:

- **Modal $250 Compute Credits:** Use Modal's cloud platform to perform heavy-lifting tasks that would be hard on local resources. For instance, you can *preprocess the entire ICD-10 dataset* using Modal's GPUs/CPUs. One idea is to generate vector embeddings for all ICD-10 code descriptions (tens of thousands of entries) using a high-quality model. Modal's credits let you spin up powerful instances to do this quickly. For example, you could use a transformer model to encode each code description into a vector for semantic search. This upfront indexing might be computationally intensive – perfect for Modal's on-demand GPUs. By doing this offline, your eventual Gradio app can load the precomputed index and respond quickly to queries. Using Modal in this way not only improves performance, it also aligns with the **Modal Choice Award** criteria – you'd be demonstrating meaningful use of their cloud platform (which could put you in the running for that $5k prize) [6].

- **Hugging Face $25 API Credits:** These credits can be applied to Hugging Face's Inference API or other services. One way to utilize them is to query hosted models for specialized tasks. For example, you might use a HuggingFace-hosted **medical NLP model** (if available) via API to complement your tool's functionality. However, since your main needs (embedding and possibly generation) can be met with either open-source models or other APIs, you might reserve the HF credits for integration testing or for image hosting if needed. Another idea: use the credits on **Hugging Face Hub datasets or AutoTrain** – for instance, you could leverage a public ICD-10 code dataset (there's one on Kaggle/HF containing all codes and descriptions [15]) and possibly fine-tune a lightweight model with AutoTrain. Given your time constraints, this might be optional, but it's worth noting you have that budget if a Hugging Face service can boost your project (and mentioning you used HF's platform in your README could help with judge impressions).

- **OpenAI & Anthropic Credits (if received):** If you were among the first 1000 sign-ups, you'll have **$25 in OpenAI credits and $25 in Anthropic** [16]. Use these to inject some top-tier AI power into your app:

- *OpenAI (GPT-3.5/4 or Embeddings):* You could call the OpenAI *embedding API* (`text-embedding-ada-002`) to vectorize text. This model was shown to be very effective for ICD coding tasks [13]. $25 in credits is plenty to embed all ICD descriptions (OpenAI embeddings are inexpensive). If you prefer not to rely on a remote API at runtime, consider using OpenAI during development to build your vector index (e.g. embed all codes via script on Modal) and then use the static index in the app.

Additionally, you might use **GPT-4** in a limited but impactful way – for example, once your tool finds the top candidate codes, you could feed the original medical description and the candidate codes into GPT-4 to *select the best code and generate a brief rationale*. This would wow judges with an explanation like, *"Code E11.40 – Type 2 diabetes with neuropathy – was chosen because the patient's description of diabetes complications matches this specific code."* Since GPT-4 is accurate in many coding scenarios [11], this adds a polished touch. Just be mindful of your token budget; $25 might handle a few hundred calls of moderate length, which should be sufficient for a demo.

- *Anthropic (Claude):* Claude is excellent for language understanding and could similarly be used to refine or validate code selections. For instance, Claude might handle longer inputs (if you had very lengthy medical notes) due to its larger context window. If you have these credits, you could experiment with Claude for summarizing a long patient note into key terms, which you then feed into your search. However, integrating both GPT-4 and Claude may be overkill; choose one for final integration to avoid complexity. The goal is to show **you can utilize state-of-the-art LLMs** to enhance accuracy.

- **Other Credits (Nebius, Hyperbolic, Mistral, SambaNova):** These are more optional but consider them if they add value:

- *Nebius $25:* Nebius is a cloud platform (with GPU offerings) – similar in use to Modal. If your Modal usage is sufficient, you may not need Nebius. But you could use Nebius AI Studio to **fine-tune a model** if desired. For example, Nebius promoted a fine-tuning credit (FINETUNE25) which could let you fine-tune a small transformer on an ICD-10 mapping task. Given the short hackathon timeline, fine-tuning a large model might be risky, but a quick fine-tune of a smaller model (like a T5 or DistilBERT) on a sample of clinical text to code mappings could potentially be done within a couple of hours on an H100 GPU. This could showcase extra effort and might improve performance on niche cases. Use this only if time permits and you feel it will significantly boost your tool's accuracy.
- *Hyperbolic $15:* Hyperbolic Labs offers affordable H100 GPU time [17]. You likely won't need this if Modal covers your GPU needs, but you could mention using Hyperbolic for some testing or to try out their hosted 405B model (they advertise a Llama 3.1 405B access [18]). For example, you might do a fun experiment: ask their large model for coding suggestions and compare. Even if it's not central to your app, noting that you tested multiple AI services shows thoroughness.
- *Mistral AI $25:* Mistral has open-source models (their 7B is public) and they offer API credits. You could use a **Mistral model** for inference as a backup or to avoid reliance on closed APIs. For instance, use the Mistral-7B or a fine-tuned variant via their API to parse medical text. If it performs well, this could be your model of choice for the live demo (ensuring your app remains functional even if OpenAI isn't used). Also, mentioning Mistral usage could put you in contention for the Mistral Choice Award (they're giving $2k in credits as a prize) [19].

- *SambaNova $25:* SambaNova might have a platform or specific models (perhaps a GPT-like model or a data stream engine). If you have access, see if they offer a pretrained model for medical text or a high-performance GPT-J style model. You could offload one task to this – e.g. use SambaNova's API to generate a summary of a complex patient note before coding. This would be another sponsor integration point (and could make you eligible for their $500 award) [20]. Only pursue this if their documentation is accessible and integration is straightforward – since you noted no prior API experience, focus on the easier wins first (OpenAI, etc.) and treat SambaNova as a bonus if time allows.

- **LlamaIndex and Retrieval Tools:** The hackathon has LlamaIndex as a sponsor, and your project naturally involves information retrieval. Consider using **LlamaIndex (GPT Index)** or a vector database to manage the ICD-10 knowledge base. For example, you can load all ICD-10 code descriptions into LlamaIndex and let it handle semantic search queries from the LLM. LlamaIndex is designed to connect LLMs with external data, which fits your use case (it can help your MCP server retrieve the most relevant codes for a given query). Using it might simplify some coding (it provides high-level APIs for adding documents and querying with an LLM under the hood) and will definitely look good to judges from LlamaIndex. Make sure to highlight in your documentation if you use it. Even if you don't, you'll likely implement similar functionality (embedding + search), which you can mention as *"a custom semantic search,"* but using the actual LlamaIndex library could save time and score points.

- **ChatGPT Plus for Development:** Although not a sponsor credit, remember you have your ChatGPT Pro subscription. Leverage GPT-4 in ChatGPT to assist you in writing code, debugging, and brainstorming throughout the hackathon. For instance, you can prompt ChatGPT for help with using the Modal API or writing a function to parse ICD code data. This will speed up development given you're newer to some of these APIs. Just be cautious to not inadvertently paste any private API keys during these interactions. Using ChatGPT for *development support* will indirectly help you maximize the value of the credits by ensuring your implementation (Modal jobs, API calls, etc.) is done efficiently and correctly on the first try.

Implementation Plan Overview

With the idea and resources in mind, here's a step-by-step plan to build the project:

1. **Gather ICD-10 Data:** Obtain a comprehensive dataset of ICD-10 codes and descriptions. The **CMS and CDC provide official files**, and there are convenient aggregated versions (e.g. a Kaggle dataset with the full 2023 ICD-10-CM code set) [15]. You can download a CSV of codes to use as your knowledge base. Since you're focusing on diagnosis codes (ICD-10-CM), that alone is ~70k entries. Load this data into your app or a database.

2. **Build the Search/Lookup Functionality:** Develop a module to search the codes by disease description or code. This has two parts:

3. *Keyword Search:* Implement a basic keyword filter (so if someone types "diabetes neuropathy", you can quickly narrow to codes whose description contains those terms). This ensures obvious matches aren't missed and provides a fallback.
4. *Semantic Similarity Search:* Use vector embeddings to find codes related to the meaning of the query. As discussed, create embeddings for each ICD description. You can do this offline using Modal + OpenAI's ada-002 or an open-source model. The medRxiv research suggests embedding-based retrieval is highly effective for this domain [13] [21]. Store vectors in a simple vector index (FAISS, Annoy, or even a list with numpy if performance is okay). At query time, embed the user's input (e.g. "Type 2 diabetes with nerve pain in feet") and find nearest code vectors. This will yield candidate codes that **may not share exact words** but are conceptually similar – a big advantage over pure keyword search.

5. Combining these, your MCP server can accept either a code (to lookup definition) or a description (to search codes). This mirrors how some existing healthcare MCP servers handle it [22] [23]. For example, if `description="diabetes with neuropathy"`, your tool returns a list of likely codes (e.g. *E11.40 Type 2 diabetes with diabetic neuropathy* as a top match).

6. **Integrate LLM for Refinement (Optional but Recommended):** Once you have a set of candidate codes from the search step, you can **boost accuracy using an LLM**:

7. Use GPT-4 (via OpenAI API) or Claude (Anthropic) to pick the best code from the top-N results. Provide the model with the original query and the top few code descriptions, and prompt it to choose which code fits best and why. Because GPT-4 is very strong at understanding nuanced language [11], it can resolve ambiguities (for example, ensuring laterality or specific complication matches the code). This addresses the scenario where multiple similar codes might come up.
8. Alternatively, use the LLM to **explain or validate** rather than select. For instance, after you determine a best match via similarity score, you could have the LLM generate a short explanation: *“The input description mentions a diabetes complication (neuropathy), which corresponds to ICD-10 code E11.40 – Type 2 diabetes with diabetic neuropathy.”* This makes your output more user-friendly and showcases AI understanding.

9. Keep this step lightweight to conserve tokens – you might only call the LLM for one or two prompts per query. This is feasible within free credit limits, especially if using GPT-3.5 for explanation (or GPT-4 sparingly). Also, if you didn’t get those API credits, you can skip this and still have a functional project; the retrieval alone is valuable. But including an LLM in the loop will make your demo shine.

10. **Build the MCP Server (Gradio App):** Develop your Gradio interface such that it can both serve as a web demo and listen for MCP requests. The hackathon organizers note *“any Gradio app can also be an MCP server/tool”* with the proper setup [1]. Concretely:

11. Use the `gradio` Python library to create a simple UI with a text box (for input description or code) and an output area (for results). This UI is mainly for judges and your video demo, so make it clean and easy: e.g. the user enters a medical condition description, and the app displays the top suggested code(s) with descriptions and perhaps the LLM-generated rationale.
12. Implement the backend logic so that it can handle requests from an AI agent via MCP. You’ll likely use the **Gradio networking API** or a lightweight server route. According to MCP standards, your tool should expose an API endpoint (or STDIO interface) that agents like Cursor or Claude Desktop can call. (The hackathon may provide a template or guidelines for making your space discoverable as an MCP tool – ensure you follow those so your submission qualifies. For example, you might tag it with “mcp-server-track” and support a standard `/api` route as shown in other projects [22].)

13. Test the MCP connectivity. If possible, use an MCP client (like the Cursor IDE or Claude’s agent mode) to connect to your running server. Verify that an agent can query something like *“Use the ICD10 tool to find codes for ‘heart attack’”* and your server responds with results. Recording this interaction will be great content for your demo video (showing an AI agent successfully calling your tool).

14. **Testing with Realistic Inputs:** Create a small set of test scenarios (from your experience or literature) to validate the tool. For example:

15. Simple case: “Type 1 diabetes mellitus” should return E10.9 (Type 1 diabetes without complications).
16. Complex case: “Patient with acute myocardial infarction (heart attack) involving left anterior descending artery” – your tool should return something like I21.02 (Anterior wall MI) or a related code.

17. Edge case: Typos or layman terms (maybe someone writes “heart attack” instead of “myocardial infarction”). If your similarity search is good, it may catch that. If not, consider adding a small thesaurus or spell-check step (you could even use an LLM to rephrase the query medically). Each test ensures your pipeline (embedding search + LLM) works as expected. Use your ChatGPT subscription to simulate some of these – you can ask GPT-4 what the correct code should be for a description and compare to your tool’s output, to gauge accuracy.

18. **Optimization and Caching:** To make the user (and judge) experience smooth, use caching where possible. For instance, if using OpenAI for embeddings at runtime (not recommended, but if you do), cache embeddings of frequent queries. Since you likely embed the entire code database once (and reuse it), that’s fine. If you call GPT for explanations, use a cache (even an in-memory dict) to avoid re-generating for the same query repeatedly during demo/testing. Also take advantage of the **Hugging Face Space hardware**: if your app needs a GPU (e.g. to run a local embedding model), consider deploying it on a GPU-backed space using your credits or the free GPU time window. But if you’ve precomputed everything and just do quick math and occasional API calls, a CPU space should suffice. The Modal approach ensures heavy work is done upfront.

19. **Polish the Documentation & Demo:** As you finish implementation, allocate time to *present* your project effectively:

20. Write a clear **README.md** in your Space. Include a description of what the tool does and why it’s useful (mention the problem of manual coding and how your AI tool helps solve it). **Cite the research or stats** to reinforce this – e.g., note that manual coding is laborious and AI can improve efficiency [10], or mention how many codes exist to justify the need for semantic search [9]. A brief architecture outline (search + GPT reasoning) will show the judges your technical approach. Also, list which sponsor resources you used (Modal, OpenAI, etc.) and express thanks – this signals you took advantage of the credits, which judges appreciate.
21. Prepare the **demo video** (as required by track guidelines [24]). In ~2-3 minutes, show the Gradio app interface: input a sample medical phrase and watch it return the ICD-10 code with an explanation. If possible, also show an AI agent using your tool (even if it’s just you simulating an agent by copying a curl command or using the provided MCP client library). The video should highlight how seamlessly your MCP server extends an LLM’s capability – e.g., a quick clip of Cursor AI auto-suggesting the code after hitting your server would be golden.
22. If there’s a **community showcase or Discord**, share your project, maybe even use your personal YouTube to talk about it. Sometimes there’s a *community choice* prize [25], so a bit of socializing can help. Plus, feedback from others can be used to refine your app in the final days.

Final Tips for Success

- **Stay Focused on Core Functionality:** With just a few days, prioritize the essential features (accurate code retrieval and agent integration). It’s tempting to add extra medical tools (like drug info, or CPT code lookup, etc.), but a well-executed single-feature tool often impresses more than a half-baked multi-tool. Quality over quantity. Your ICD-10 assistant, done properly, can be very compelling.

- **Use Your Strengths:** As a network engineer/developer, you excel at backend logic. Lean into that by making your server robust and efficient. For example, implement solid error handling (if the user input is unclear, maybe return a message asking for clarification – an LLM could even generate that). Your Linux skills will help in environment setup and maybe writing a quick script to parse the ICD data file. These "under the hood" skills will show through in the stability of your demo.

- **Learn the APIs with Minimal Overhead:** Since you're new to using APIs, start with the most straightforward ones. The OpenAI Python SDK is very easy to use – a few lines with `openai.Embedding.create()` or `openai.ChatCompletion.create()` and you're set (the documentation has clear examples). Similarly, Modal provides a Pythonic interface to run functions on the cloud; you can follow their quickstart to offload tasks. Don't get bogged down in too many new services – use the ones that integrate cleanly with Python. The sponsors provided these credits to remove resource barriers, so take advantage of them, and don't be afraid to look at examples or ask in the hackathon forum if you hit a snag. Time is short, but the community and resources are there to help.

- **Validation with Domain Knowledge:** While you may not be a medical expert, try to incorporate some domain validation. For instance, you know ICD-10 has hierarchical code structure (e.g., codes starting with "E11" are all type 2 diabetes). Simple checks like ensuring the code and description truly match the query context can be implemented by rules or by the LLM. This will prevent obvious mistakes (choosing a code that's related but not actually what the description means). Even a quick sanity-check prompt to GPT-3.5 like "Does this code describe the condition well? Yes/No" could filter out bad answers. Little touches like that show you thought about accuracy, which judges will note.

- **Highlight the Use of Credits/Tech in Submission:** Explicitly mention in your README or presentation how you utilized the provided resources. For example: *"This project was built using Modal's cloud GPUs to preprocess data, OpenAI's GPT-4 for code validation (using hackathon API credits), and LlamaIndex for efficient retrieval."* This not only gives credit to the sponsors (good etiquette) but also reinforces that you made the most of the hackathon's offerings – a trait of a resourceful hacker.

By following this strategy, you'll deliver a project that **showcases AI solving a real problem, uses cutting-edge methods, and capitalizes on the hackathon's free resources**. A well-executed ICD-10 MCP server can absolutely contend for the top spot in the track. Good luck, and enjoy the process of building something impactful!

Sources:

- Hackathon official guidelines and track description [1] [26]
- Research on AI for ICD-10 coding accuracy and methods [13] [11]
- Example of ICD-10 integration in an MCP healthcare server [14]
- Data availability for ICD-10 codes (2023 CMS dataset) [15]
- Nature overview of clinical coding complexity (need for AI) [8] [9]

[1] [2] [3] [4] [5] [6] [16] [19] [20] [24] [25] [26] Agents-MCP-Hackathon (Agents-MCP-Hackathon)
https://huggingface.co/Agents-MCP-Hackathon

[7] [8] [9] Automated clinical coding: what, why, and where we are? | npj Digital Medicine
https://www.nature.com/articles/s41746-022-00705-7?error=cookies_not_supported&code=690add16-0abe-4317-971f-c6b5cca063b2

[10] [11] [12] Frontiers | AI integration in nephrology: evaluating ChatGPT for accurate ICD-10 documentation and coding
https://www.frontiersin.org/journals/artificial-intelligence/articles/10.3389/frai.2024.1457586/full

[13] [21] Medical Diagnosis Coding Automation: Similarity Search vs. Generative AI | medRxiv
https://www.medrxiv.org/content/10.1101/2024.04.26.24306470v1.full

[14] [22] [23] GitHub - Cicatriiz/healthcare-mcp-public: A Model Context Protocol server providing AI assistants with access to healthcare data tools, including FDA drug information, PubMed research, health topics, clinical trials, and medical terminology lookup.
https://github.com/Cicatriiz/healthcare-mcp-public

[15] ICD-10-CM Codeset 2023 - Kaggle
https://www.kaggle.com/datasets/mrhell/icd10cm-codeset-2023

[17] Highlights by Hyperbolic (@hyperbolic_labs) / X
https://twitter.com/hyperbolic_labs/highlights

[18] Access Llama 3.1 405B: Model and API for FREE - Hyperbolic
https://hyperbolic.xyz/blog/llama-3-1-405b-support